US Army Corps
of Engineers
Hydrologic Engineering Center

HEC-DSSVue

HEC Data Storage System Visual Utility Engine

User's Manual

Version 1.2
May 2005

Approved for Public Release – Distribution Unlimited **CPD-79**

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the date needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** May 2005	**3. REPORT TYPE AND DATES COVERED** Computer Program Document No. 79

4. TITLE AND SUBTITLE HEC-DSSVue HEC Data Storage System Visual Utility Engine	**5. FUNDING NUMBERS**
6. AUTHOR(S) William J. Charley	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Corps of Engineers Institute for Water Resources Hydrologic Engineering Center (HEC) 609 Second Street Davis, CA 95616-4687	**8. PERFORMING ORGANIZATION REPORT NUMBER** CPD-79
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release. Distribution of this document is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

THE U.S. Army Corps of engineer's Hydrologic Engineering Center Data Storage System, or HEC-DSS, is a database system designed to efficiently store and retrieve scientific data that is typically sequential. Such data types include, but are not limited to, time series data, curve data, spatial-oriented gridded data, textual data (such as this manual), and others. The system was designed to make it easy for users and application programs to retrieve and store data.

HEC-DSSVue (HEC-DSS Visual Utility Engine) is a graphical user interface program for viewing, editing, and manipulating data in HEC-DSS database files.

14. SUBJECT TERMS HEC-DSS, HEC-DSSVue, Data Storage System, database, computer program	**15. NUMBER OF PAGES** 316
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UNLIMITED

HEC-DSSVue

HEC Data Storage System Visual Utility Engine

User's Manual

Version 1.2
May 2005

US Army Corps of Engineers
Institute for Water Resources
Hydrologic Engineering Center
609 Second Street
Davis, CA 95616

(530) 756-1104
(530) 756-8250 FAX
www.hec.usace.army.mil

HEC Data Storage System, HEC-DSS
Software Distribution and Availability Statement

The HEC-DSS and HEC-DSS Vue executable code and documentation are public domain software that were developed by the Hydrologic Engineering Center for the U.S. Army Corps of Engineers. The software was developed at the expense of the United States Federal Government, and is therefore in the public domain. This software can be downloaded for free from our internet site (www.hec.usace.army.mil). HEC cannot provide technical support for this software to non-Corps users. See our software vendor list (on our web page) to locate organizations that provide the program, documentation, and support services for a fee. However, we will respond to all documented instances of program errors. Documented errors are bugs in the software due to programming mistakes not model problems due to user-entered data.

Table of Contents

List of Figures

List of Tables

List of Examples

Forward

The U.S. Army Corps of Engineers' Hydrologic Engineering Center Data Storage System, or HEC-DSS, is a database system designed to efficiently store and retrieve scientific data that is typically sequential. Such data types include, but are not limited to, time series data, curve data, spatial-oriented gridded data, textual data (such as this manual), and others. The system was designed to make it easy for users and application programs to retrieve and store data. HEC-DSSVue (HEC-DSS Visual Utility Engine) is a graphical user interface program for viewing, editing, and manipulating data in HEC-DSS database files.

HEC-DSS originated at the Hydrologic Engineering Center in 1979 under the direction of Dr. Art Pabst. Since that time, many have worked on the development of the HEC-DSS software and the HEC-DSS utility programs, including William Charley, Al Montalvo, Carl Franke, Paul Ely, Robert Carl, Dennis Huff, and numerous others.

Mr. William Charley, leader of the software development team, designed HEC-DSSVue and created the main interface screen. Resource Management Associates (RMA) of Fairfield, California, under the direction of Dr. John DeGeorge, aided in the development of the user interface. The RMA staff included Richard Rachiele, who translated the DSSMATH functions into Java, Mark Ackerman wrote the math function screens, and Shannon Newbold was responsible for the graphics. Dr. Cassie Carter, under contract with RMA, wrote the initial draft of this manual. Mr. Darryl Davis was the Director of HEC during the development of this software.

Chapter 1

Introduction

HEC-DSSVue (Hydrologic Engineering Center Data Storage System Visual Utility Engine) is a graphical user interface program for viewing, editing, and manipulating data in HEC-DSS database files. With HEC-DSSVue, you may plot, tabulate, and edit data, as well as manipulate data with over fifty mathematical functions. Along with these functions, HEC-DSSVue provides several utility functions that allow you to enter data sets into a database, rename data set names, copy data sets to other HEC-DSS database files, and delete data sets.

Typically, you will select data sets from a sorted / filtered list of names in a HEC-DSS database file using a mouse. HEC-DSSVue also incorporates the "Jython" standard scripting language, which allows you to specify a routine sequence of steps in a text format, and then execute the sequence from a user-defined button or from a "batch" process.

HEC-DSSVue was written using the Java programming language, which allows it to be run under a variety of different operating systems. Fully supported systems include Microsoft Windows 98 / ME / NT / 2000 / XP, and Sun Solaris (Unix). HEC-DSSVue has been installed on RedHat Linux and other systems. Contact HEC for the availability of these.

1.1 Overview of the HEC Data Storage System

The HEC Data Storage System, or HEC-DSS, is a database system designed to efficiently store and retrieve scientific data that is typically sequential. Such data types include, but are not limited to, time series data, curve data, spatial-oriented gridded data, textual data (such as this manual), and others. The system was designed to make it easy for users and application programs to retrieve and store data.

1.1.1 Background

HEC-DSS was the result of a need that emerged in the late 1970s. During that time most hydrologic studies were performed in a step-wise fashion, passing data from one analysis program to another in a manual mode. While this was functional, it was not very productive. Programs that used the same type of data, or that were sequentially related, did not use a common data format. Also, each program had to have its own set of graphics routines, or other such functions, to aid in the program's use.

The Kissimmee River study, performed by HEC for the Jacksonville District beginning in 1978, required an orderly approach to properly manage the study data and the analysis results. A large number of alternative plans and conditions were to be processed in this project. This study gave birth to the first version of HEC-DSS. The basic design provided for the storage of data in a standard form, independent of any particular program. The data would be provided to the programs when it was needed, and results would be stored in the same independent form for use by utilities and other applications programs. The early design of HEC-DSS was conceived to support files containing many hundreds of data sets, or even as many as a few thousand. As the use of HEC-DSS expanded into real-time data storage applications, data files were written to manage tens and hundreds of thousands of data sets. The current HEC-DSS version is designed for rapid storage and retrieval of data sets from files containing as few as 40 to 50 records to files containing more than 100,000 records.

1.1.2 HEC-DSS Contrasted with Other Database Systems

HEC-DSS is designed to be optimal for storing and retrieving large sets, or series, of data. HEC-DSS incorporates a modified hashing algorithm and hierarchical design for database accesses. This algorithm provides quick access to data sets and an efficient means of adding new data sets to the database. Additionally, HEC-DSS provides a flexible set of utility programs, and it is easy to add to a user's application program. These are the features that distinguish HEC-DSS from most commercial database programs and make it optimal for scientific applications.

HEC-DSS is designed specifically for the storage and retrieval of large sets, or series, of data. This includes daily flow values, hourly precipitation measurements, rating tables, and pages of text information. HEC-DSS is not optimized for dealing with small data sets or single data values, nor is it effective at conditional data searches common to relational database systems. In contrast, most commercial databases are designed for small sets of data, or elements. Such elemental data includes employee records, accounting data, and inventory of stock.

Commercial databases usually employ a relational model in which data is stored in a related manner. A relational database can be viewed essentially as a collection of tables (composed of rows and columns). This type of system requires the construction of a data definition or data dictionary file. Although a relational database requires some initial setup, it can effectively store short data sets comprised of both characters and numbers. Relational database systems use the ANSI ratified Structured Query Language (SQL) to access data.

While relational databases are ideal for elemental data sets, they are not as practical for longer series of data. HEC-DSS, however, is designed for such sets of data. HEC-DSS database files are not defined by a data definition file like the relational model requires. Thus, there is no set up required by the user. (HEC-DSS data is defined by the pathname and conventions used.) The type of data generally stored in HEC-DSS does not lend itself well to a query language such as SQL (although the selective catalog feature has some similar capabilities).

Also unlike many commercial database systems, the HEC-DSS was designed to be easily added to a user's application program. In traditional "C" and Fortran programs, only two or three function calls are needed to access data. Those calls identify the database, the pathname, and a time window (if desired). Besides these languages, an extensive set of classes are available in both C++ and Java languages (including some access through Visual Basic), the languages with which most new HEC "NexGen" programs are being developed. The NexGen programs make extensive use of HEC-DSS. However, these programs are designed so the interaction with HEC-DSS is seamless; most of the time users will not know that it is being used.

1.2 General Concepts for HEC-DSS

HEC-DSS uses a block of sequential data as the basic unit of storage. This concept results in a more efficient access of time series or other sequentially related data. Each block contains a series of values of a single variable over a time span appropriate for most applications. The basic concept underlying HEC-DSS is the organization of data into records of continuous, applications-related elements, as opposed to individually addressable data items. This approach is more efficient for scientific applications than a conventional database system because it avoids the processing and storage overhead required to assemble an equivalent record from a conventional system.

Data is stored in blocks, or records, within a file, and each record is identified by a unique name called a "pathname." Each time data is stored or retrieved from the file, its pathname must be given. Along with the data, information about the data (*e.g.*, units) is stored in a "header array." HEC-DSS automatically stores the name of the program writing the data, the number of times the data has been written to, and the last written date and time. HEC-DSS documents stored data completely via information contained in the pathname and stored in the header, so no additional information is required to identify it. The self-documenting nature of the database allows information to be recognized and understood months or years after it was stored.

The pathname is the key to the data's location in the database. HEC-DSS analyzes each pathname to determine a "hash" index number. This index determines where the data set is stored within the database. The design ensures that very few disk accesses are made to retrieve or store data sets. One data set is not directly related to another, so there is no need to update other areas of the database when a new data set is stored.

Because of the self-documenting nature of the pathname and the conventions adopted, there is no need for a data dictionary or data definition file as required with other database systems. In fact, there are no database creation tasks or any database setup. Both HEC-DSS utility programs and applications that use HEC-DSS will generate and configure HEC-DSS database files automatically. There is no pre-allocation of space; the software automatically expands the file size as needed.

A HEC-DSS database file has a user-specified conventional name, with an extension of “.dss.” As many database files as desired may be generated and there are no size limitations, apart from available disk space. Corps offices have HEC-DSS files that range from a few data sets to many thousands. HEC-DSS adjusts internal tables and hash algorithms to match the database size so as to access both small and very large databases efficiently.

HEC-DSS database files are “direct-access” binary files with no published format. Only programs linked with the HEC-DSS software library can be used to access HEC-DSS files. Direct access files allow efficient retrieval and storage of blocks of data compared to sequential files.

A principal feature of HEC-DSS is that many users can read and write data to a single database at the same time. This multi-user access capability is implemented with system record locking and flushing functions. There is no daemon or other background program managing accesses to a database. A database may exist on a Windows or Unix server machine, and can be accessed by users on PC’s or other computers via NFS or the Microsoft network, as long as locking and flushing functions are implemented.

1.2.1 Pathnames

HEC-DSS references data sets, or records, by their *pathnames*. A pathname may consist of up to 391 characters and is, by convention, separated into six parts, which may be up to 64 characters each. Pathnames are automatically translated into all upper case characters. They are separated into six parts (delimited by slashes "/") labeled "A" through "F,” as follows:

/A/B/C/D/E/F/

For regular-interval time series data, the part naming convention is:

Part	Description
A	Project, river, or basin name
B	Location
C	Data parameter
D	Starting date of block, in a 9 character military format
E	Time interval
F	Additional user-defined descriptive information

A typical regular-interval time series might be:

/RED RIVER/BEND MARINA/FLOW/01JAN1995/1DAY/OBS/

1.2.2 Catalogs

HEC-DSS utility programs, including HEC-DSSVue, will generate a list of the pathnames in a HEC-DSS file and store that list in a "catalog" file. The catalog file is a list of the record pathnames in the file, along with their last written date and time and the name of the program that wrote that record. The catalog is usually sorted alphabetically by pathname parts. Each pathname has a record tag and a reference number, either of which may be used in place of the pathname in several of the utility programs. The name given to the catalog file is the HEC-DSS file's name with an extension of ".dsc."

A special catalog file, the "condensed catalog," is useful mainly for time series data. In this type of catalog, pathname parts display in columns, and pathnames for time series data, differing only by the date (D part), are referenced with one line.

1.2.3 Data Conventions

HEC-DSS can store data of most any type using a pathname of any structure. To facilitate the ability of application and utility programs to work with and display data, standard record conventions were developed. These conventions define what should be contained in a pathname, how data is stored and what additional information is stored along with the data. For regular-interval time series data (*e.g.*, hourly data), the conventions specify that data is stored in blocks of a standard length, uniform for that time interval, with a pathname that contains the date of the beginning of the block and the time interval. The conventions identify how a pathname for the data should be constructed. Conventions have been defined for regular and irregular interval time series data, paired (curve) data, gridded data (such as NEXRAD radar data) and text (alphanumeric) data.

Regular-interval time series data is data that occurs at a standard time interval. This data is divided into blocks whose length depends on the time interval (for example, hourly data is stored with a block length of a month, while daily data is stored with a block length of a year). Only the date and time of the first piece of data for a block is stored; the times of the other data elements are implied by their location within the block. If a data element, or a set of elements, does not exist for a particular time, a missing data flag is placed in that element's location. Data quality flags may optionally be stored along with a regular-interval time series record.

Irregular-interval time series data does not have a constant time interval between values. This type of data is stored with a date/time stamp for each element. The user-selectable block size is based on the amount of data that is to be stored. For example, the user may select a block length of a month or a year. Because a date/time stamp is stored with each data element, approximately twice the amount of space is required compared with regular-interval time series data. Data quality flags may optionally be stored along with an irregular-interval time series record.

A convention for paired data has been defined for data that generally defines a curve. It is used for rating tables, flow-frequency curves, and stage-damage curves, etc. One paired data record may contain several curves within it, as long as it has a common set of ordinates. For example a stage-damage curve will contain a set of stages, and it may have associated residential damages, commercial damages, agricultural damages, etc. However, a stage-damage curve and a stage-flow curve cannot be stored in the same record.

Conventions for spatially gridded data can be found in the "GridUtil" User's Manual. Text conventions are specified in the HEC-DSS Programmer's Manual.

1.2.4 Database File Size

HEC-DSS Version 6 database files have been designed for, and tested to, a 2 Gigabyte file size. At the time of the preparation of this manual, modifications were underway to expand Version 6 files to a limit of 16 Gigabytes. These modifications do not modify the file structure.

Chapter 2

Installing HEC-DSSVue

This chapter describes the minimum and recommended computer system requirements for running the program, explains how to install and uninstall the software, and gives system administrators information on permissions that are changed during setup.

2.1 Operating Systems Supported

The program is multi-platform capable and has been produced for the following computer operating systems:

- Microsoft Windows 98/ME
- Microsoft Windows NT 4.0 with Service Pack 5 or later
- Microsoft Windows 2000
- Microsoft Windows XP
- Sun Solaris 2.5 or later
- Linux*

* Consult the HEC web site for more information about specific Linux distributions that HEC-DSSVue has been tested on. This list will be updated to reflect any additional distributions that are tested.

2.2 Microsoft Windows Installation

To install HEC-DSSVue on a computer running Microsoft Windows, ensure that your computer meets the minimum hardware requirements and follow the instructions below to run the Setup program.

2.2.1 Hardware Requirements

For optimum performance, make sure that your computer has at least the minimum required hardware before you install the HEC-DSSVue software. This version of HEC-DSSVue will run on a PC that has the following:

- Intel® Pentium® II processor (or equivalent), 500 MHz or higher
- 60 MB available hard disk storage
- A minimum of 256 MB of RAM

2.2.2 Installation Procedure

Before installing HEC-DSSVue, make sure you have local administrator rights. Many home users already have administrator rights by default. If you're in a corporate environment however, you may need to ask your system administrator to install HEC-DSSVue for you.

Do the following to install HEC-DSSVue:

1. Download HEC-DSSVue setup from the HEC web site http://www.hec.usace.army.mil
2. To start the installation, double-click on the setup file after it downloads.
3. Complete the InstallShield Wizard to install HEC-DSSVue.
4. You will not have to restart your computer unless the Microsoft Installer program requires an update.

2.2.3 Uninstall Procedure

Follow this procedure to remove HEC-DSSVue from your computer.

1. Close HEC-DSSVue.
2. From the Windows Start Menu, select **Settings-Control Panel-Add/Remove Programs**. Select HEC-DSSVue from the list of programs and click Remove. Windows XP users will just go to **Control Panel-Add/Remove Programs** from the Start Menu.
3. The Uninstall program prompts you to confirm that you want to remove the application. Click **Yes**.
4. When uninstall is completed, close Add/Remove Programs and the Windows Control Panel.

2.2.4 Managed Install Permissions

The following summarizes the required permissions for HEC-DSSVue to run properly.

In the root of where HEC-DSSVue is installed:

1. Directory **HecDssVue\scripts** has the following permissions:
    a. Administrators (group) – Full Control
    b. Users (group) – Full Control
    c. SYSTEM – Full Control

2. File **HecDssVue.cmd** has the following permissions:
   a. Administrators (group) – Full Control
   b. Users (group) – Full Control
   c. SYSTEM – Full Control

3. In the Registry the following is created if it does not exist already:
   a. HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Prefs

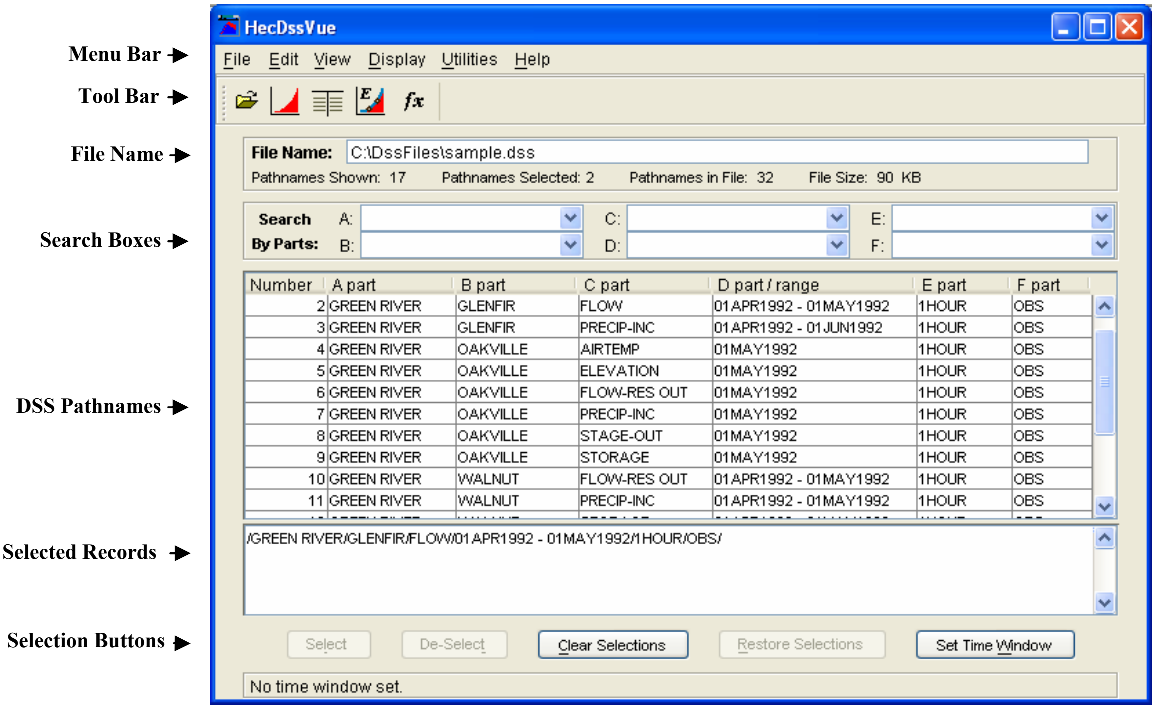
2.3 Unix and Linux Installation

Setup instructions for the Unix/Linux platforms are available on the HEC-DSSVue download page.

Chapter 3

Using HEC-DSSVue: An Overview

With **HEC-DSSVue** you can access, visualize, and manipulate data stored in HEC-DSS database files using a variety of utilities and functions available from the main HEC-DSSVue screen, shown in Figure 3.1.



File Name: C:\DssFiles\sample.dss
Pathnames Shown: 17 Pathnames Selected: 2 Pathnames in File: 32 File Size: 90 KB

Search By Parts: A: B: C: D: E: F:

Number	A part	B part	C part	D part / range	E part	F part
2	GREEN RIVER	GLENFIR	FLOW	01APR1992 - 01MAY1992	1HOUR	OBS
3	GREEN RIVER	GLENFIR	PRECIP-INC	01APR1992 - 01JUN1992	1HOUR	OBS
4	GREEN RIVER	OAKVILLE	AIRTEMP	01MAY1992	1HOUR	OBS
5	GREEN RIVER	OAKVILLE	ELEVATION	01MAY1992	1HOUR	OBS
6	GREEN RIVER	OAKVILLE	FLOW-RES OUT	01MAY1992	1HOUR	OBS
7	GREEN RIVER	OAKVILLE	PRECIP-INC	01MAY1992	1HOUR	OBS
8	GREEN RIVER	OAKVILLE	STAGE-OUT	01MAY1992	1HOUR	OBS
9	GREEN RIVER	OAKVILLE	STORAGE	01MAY1992	1HOUR	OBS
10	GREEN RIVER	WALNUT	FLOW-RES OUT	01APR1992 - 01MAY1992	1HOUR	OBS
11	GREEN RIVER	WALNUT	PRECIP-INC	01APR1992 - 01MAY1992	1HOUR	OBS

/GREEN RIVER/GLENFIR/FLOW/01APR1992 - 01MAY1992/1HOUR/OBS/

No time window set.

Figure 3.1 HEC-DSSVue Main Window

This chapter provides an overview of HEC-DSSVue and its basic operations using the main menu and selection buttons.

3.1 HEC-DSSVue Main Screen

The main screen of HEC-DSSVue consists of a **Menu Bar**, **Tool Bar**, **File Name** box, **Search boxes**, a list of HEC-DSS **Pathnames,** a list of **Selected Pathnames**, and **Selection Buttons**. The following sections describe these features in detail.

3.1.1 Menu Bar

Main menu options in HEC-DSSVue (Figure 3.2) allow you to search for, select, and edit HEC-DSS data sets; control the display of pathnames; and access plots and tables, among other tasks. The HEC-DSSVue menus are as follows:

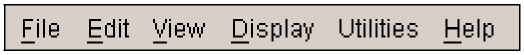


Figure 3.2 HEC-DSSVue Menu

File File menu commands are **New**, **Open, Close DSS File(s)**, **Print Catalog Preview**, **Print Catalog** and **Exit**. The file menu also lists the last 6 most recently used files.

Edit Edit menu commands are **Tabular Edit…**, **Graphical Edit...**, and **Select All**.

View The View menu allows you to customize the display of HEC-DSS pathnames, refresh the catalog and search pathnames. Available commands are **Pathname List**, **Pathname Parts**, **Condensed Catalog**, **No Pathnames**, **Refresh Catalog, Search pathnames by string**, and **Search pathnames by parts**.

Display Use the Display menu to open plots and tables with the **Plot** and **Tabulate** commands. You may also set the **Plot Data Options** of **Normalize**, and **Synch data set times to first**.

Utilities Utilities menu provides access to **Math Functions**, **Manual Data Entry** (Time Series and Paired Data), and it allows you to **Rename Records**, **Duplicate Records**, **Copy Records**, **Delete Records**, **Undelete Records**, **Merge Files**, **Squeeze**, access the Script Button Frame with the **Script Browser** command, access the **Script Selector**, and view HEC-DSS **Status**.

3.1.2 Tool Bar

Tool bar buttons provide shortcuts to frequently used menu commands:

Opens a **File Browser**, which allows you to select a HEC-DSS file (same as **Open** in the **File** menu).

Plots the selected data (same as **Plot** in the **Display** menu)

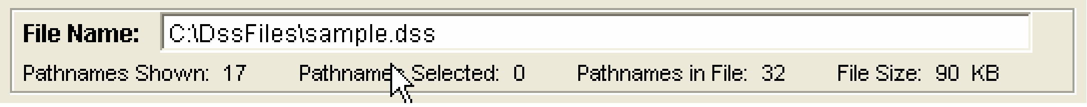
Displays the selected data in table form (same as **Tabulate** in the **Display** menu).

Graphically Edit the selected data (same as **Graphical Edit** in the **Edit** menu).

Open the **Math Functions** dialog with the selected data.

3.1.3 File Name

The **File Name** box (Figure 3.3) displays the file name and location of the currently open HEC-DSS file. Also, if you know the exact location and name of the HEC-DSS database file you wish to open, you can type it directly into the File Name box to open it, then press the Enter key.



File Name: C:\DssFiles\sample.dss

Pathnames Shown: 17 Pathnames Selected: 0 Pathnames in File: 32 File Size: 90 KB

Figure 3.3 File Name Box

Beneath the file name are the number of pathnames in the list, the number selected, the total number in the database file, and the size of the database file. If you do not have access to write to the file, the words "Read Only" will be displayed following the file size.

3.1.4 Search Boxes

The Data Selection List also provides a search area where you can **Search Pathnames** (Figure 3.4) or **Search Parts** (Figure 3.5) of an open HEC-DSS file. The search method desired is set from the **View** menu.

To search and display all pathnames containing a specific string, type that string into the box and press the **Search** button. Those pathnames in the file that contain that string will be displayed in the list.

The **Search by Parts** option provides a selectable drop down list of the parts of all the pathnames in the file. When a part is selected, only those pathnames that match that part are shown.

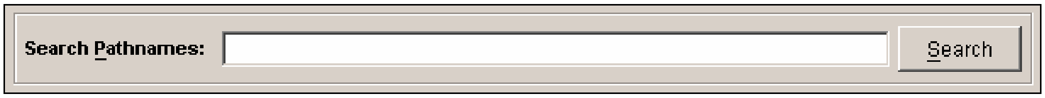


Figure 3.4 Search Pathnames option

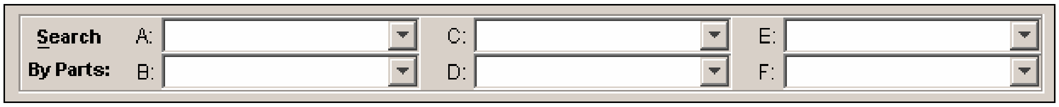
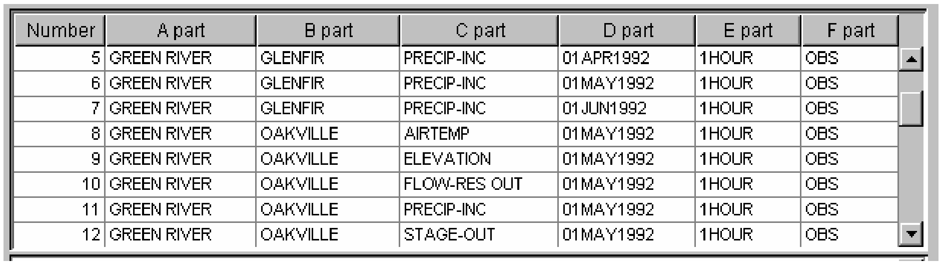


Figure 3.5 Search by Parts option

3.1.5 List of HEC-DSS Pathnames

Once you have opened an HEC-DSS file, its pathnames appear in a list beneath the search area, either as a *Pathname List*, or by *Pathname Parts*, in *Condensed Catalog* format, or with *No Pathnames* displayed. (The No Pathnames option is typically used to access very large databases using only scripted functions.) These options are set from the View menu, as described in Section 3.4.1. Figure 3.6 shows the pathname parts displayed.

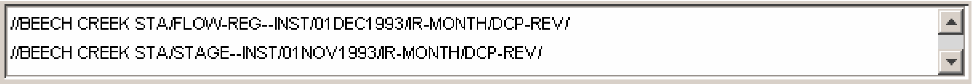


Number	A part	B part	C part	D part	E part	F part
5	GREEN RIVER	GLENFIR	PRECIP-INC	01APR1992	1HOUR	OBS
6	GREEN RIVER	GLENFIR	PRECIP-INC	01MAY1992	1HOUR	OBS
7	GREEN RIVER	GLENFIR	PRECIP-INC	01JUN1992	1HOUR	OBS
8	GREEN RIVER	OAKVILLE	AIRTEMP	01MAY1992	1HOUR	OBS
9	GREEN RIVER	OAKVILLE	ELEVATION	01MAY1992	1HOUR	OBS
10	GREEN RIVER	OAKVILLE	FLOW-RES OUT	01MAY1992	1HOUR	OBS
11	GREEN RIVER	OAKVILLE	PRECIP-INC	01MAY1992	1HOUR	OBS
12	GREEN RIVER	OAKVILLE	STAGE-OUT	01MAY1992	1HOUR	OBS

Figure 3.6 List of HEC-DSS Pathnames by Parts

3.1.6 Selected Pathnames

When you select a pathname (see Section 3.5), it appears beneath the list of all pathnames in the selection area (Figure 3.7). You can make the size of selected list taller or shorter by grabbing the separator line between the two lists with the mouse, and moving it up or down.



//BEECH CREEK STA/FLOW-REG--INST/01DEC1993/IR-MONTH/DCP-REV/
//BEECH CREEK STA/STAGE--INST/01NOV1993/IR-MONTH/DCP-REV/

Figure 3.7 Selected Pathnames

3.1.7 Selection Buttons

There are five buttons at the bottom of the Data Selection List window (Figure 3.8).

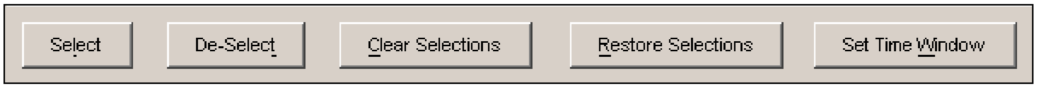


Figure 3.8 Select, De-Select, Clear Selections, Restore Selections, and Set Time Window Buttons

- **Select:** You add a pathname to the selection list by highlighting it and clicking the **Select** button. Until you highlight a pathname, the Select button remains inactive. You can also do a "quick select," if the selection list is empty, by highlighting a pathname, then pressing the plot or tabulate button.
- **De-Select:** To remove a pathname from the selection list, you highlight the pathname in the lower list then press the **De-Select** button.
- **Clear Selections:** You can remove all pathnames from the selection list by clicking the **Clear Selections** button.
- **Restore Selections:** The **Restore Selections** button restores all selections you have cleared or de-selected.
- **Set Time Window:** You can set the time window of the data to view with the **Set Time Window** button.

3.2 File Menu Operations

HEC-DSSVue's **File** menu allows you to create a **New** HEC-DSS file, **Open** an existing HEC-DSS file, **Close** the HEC-DSS file you have opened**,** show the **Print Catalog Preview, Print the Catalog,** open a recently opened file, and **Exit** the program. The following sections discuss these operations.

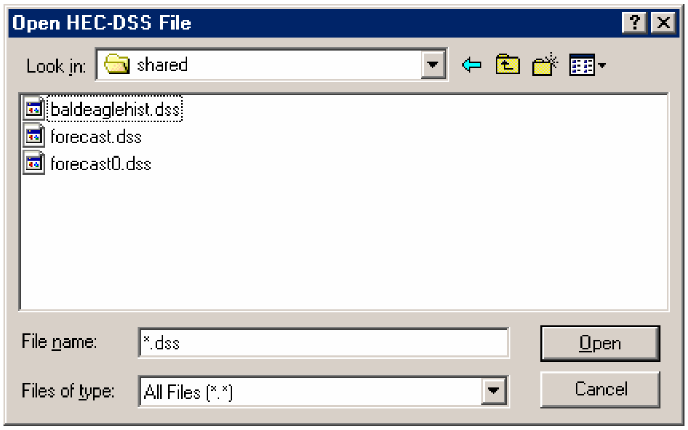
3.2.1 Creating a New HEC-DSS File

To create a new HEC-DSS database file, from the **File** menu, click **New**. When the **File** dialog box opens, type in a name for the new file and HEC-DSSVue will create it for you. Data can then be entered through the **Manual Data Entry** window from the **Utilities** menu. If you copy records from an existing file, the new DSS file does not need to exist; it will be created automatically.

3.2.2 Opening a HEC-DSS File

If you know the name of the HEC-DSS database file you wish to open, you can type the file name (including the path) directly into the **File Name** box (Figure 3.3). Otherwise, choose **Open** from the **File** menu or click the 📂 button to select the HEC-DSS database file you want.

A **File Browser** window will open, as shown in Figure 3.9.



Look in: shared

File name: *.dss

Files of type: All Files (*.*)

Figure 3.9 Open HEC-DSS File

In the **File Browser** window, use the standard Windows controls to browse to the HEC-DSS file that you wish to open, then click **Open**.

The name of the file you have selected will now appear in the **File Name** box (Figure 3.3), and the pathnames of the records contained in the file will display in the **List of HEC-DSS Pathnames**.

3.2.3 Closing the HEC-DSS File

To close the HEC-DSS database file that you currently have opened, from the **File** menu, click **Close**. This will allow some older MS-DOS based programs to access the file, or for you to rename or delete the file. Note: HEC-DSS is a multi-user database system. Some older MS-DOS programs exclusively locked HEC-DSS files to fully utilize disk caching.

3.2.4 Printing the Catalog

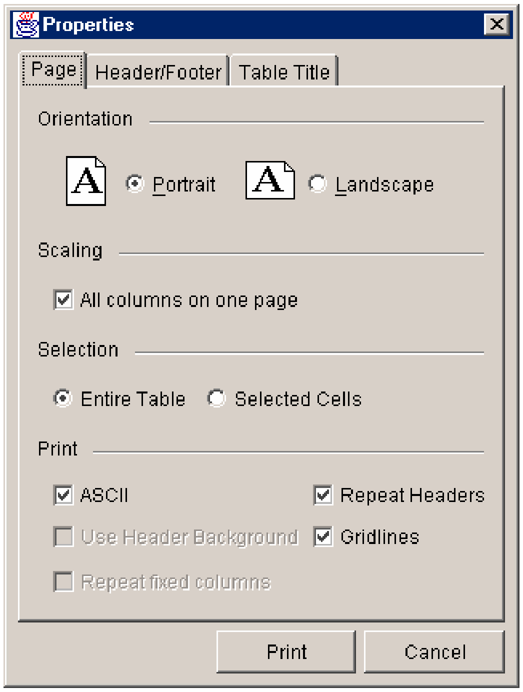
You can Print the catalog or show what the catalog will look like when it is printed, from the **Print Catalog Preview…** command or from the **Print Catalog** command from the **File** menu. Each command opens the **Print Properties** dialog box (Figure 3.10), which offers options on three tabs.

The **Page** tab allows you to specify the page Orientation, Scaling, and Selection; you can also choose to print the table as ASCII, Repeat Headers on every page, and print the Gridlines.

On the **Header/Footer** tab, you can type in the header and footer you want to appear on your printed pages.

The **Table Title** tab offers a default title for the table based on the data source. You may edit this title.

On the **Print Properties** dialog box, the **Print** button performs two functions, depending on whether you arrived at the dialog box via the **Print** command or the **Print Preview** command.



Orientation

● Portrait ○ Landscape

Scaling

☑ All columns on one page

Selection

● Entire Table ○ Selected Cells

Print

☑ ASCII ☑ Repeat Headers

☐ Use Header Background ☑ Gridlines

☐ Repeat fixed columns

Figure 3.10 Print Properties Dialog Box

3.2.5 Exiting HEC-DSSVue

To exit HEC-DSSVue, from the **File** menu, click **Exit**. (If you have opened HEC-DSSVue from another application, such as CWMS or ResSim, select **Close** from HEC-DSSVue's **File** menu.)

3.3 Edit Menu Operations

Edit menu commands are **Tabular Edit, Graphical Edit**, and **Select All**.

3.3.1 Tabular Editing of HEC-DSS Data

To edit data in a tabular format, from the **Edit** menu, click **Tabular Edit**. This will display the data in a table format with editing turned on and is the same as selecting **Allow Editing** from a tables **Edit** menu.

Chapter 4 on **Utilities** discusses tabular editing in detail.

3.3.2 Graphical Editing of HEC-DSS Data

To edit data in a graphical format, from the **Edit** menu, click **Graphical Edit**. This will display the data in a combined plot/table format.

Chapter 4 on **Utilities** discusses graphical editing in detail.

3.3.3 Selecting All Pathnames in a HEC-DSS File

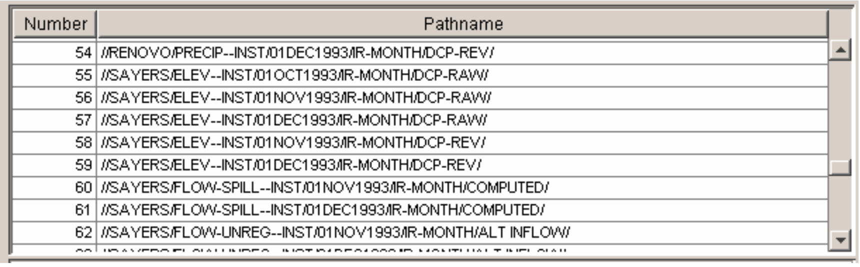
From the **View** menu, you can select all pathnames in a HEC-DSS file for visualization by clicking **Select All** . Refer to Section 3.5 for more information about selecting pathnames for visualization.

3.4 View Menu Operations

The **View** menu allows you to customize the display of HEC-DSS pathnames and search pathnames. The following sections describe these options.

3.4.1 Choosing a Display Mode for HEC-DSS Pathnames

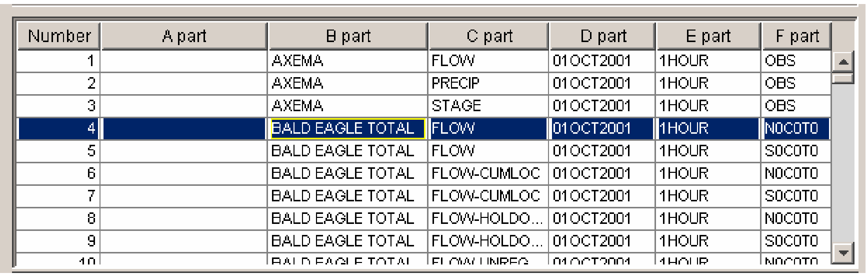
To specify the display mode for HEC-DSS Pathnames, from the **View** menu, choose **Pathname List**, **Pathname Parts**, **Condensed Catalog**, or **No Pathnames**. Figure 3.11shows a *Pathname List* of HEC-DSS files displayed.



Number	Pathname
54	//RENOVO/PRECIP--INST/01DEC1993/IR-MONTH/DCP-REV/
55	//SAYERS/ELEV--INST/01OCT1993/IR-MONTH/DCP-RAW/
56	//SAYERS/ELEV--INST/01NOV1993/IR-MONTH/DCP-RAW/
57	//SAYERS/ELEV--INST/01DEC1993/IR-MONTH/DCP-RAW/
58	//SAYERS/ELEV--INST/01NOV1993/IR-MONTH/DCP-REV/
59	//SAYERS/ELEV--INST/01DEC1993/IR-MONTH/DCP-REV/
60	//SAYERS/FLOW-SPILL--INST/01NOV1993/IR-MONTH/COMPUTED/
61	//SAYERS/FLOW-SPILL--INST/01DEC1993/IR-MONTH/COMPUTED/
62	//SAYERS/FLOW-UNREG--INST/01NOV1993/IR-MONTH/ALT INFLOW/
63	//SAYERS/FLOW-UNREG--INST/01DEC1993/IR-MONTH/ALT INFLOW/

Figure 3.11 HEC-DSSVue – Data Selection List window, pathname list displayed

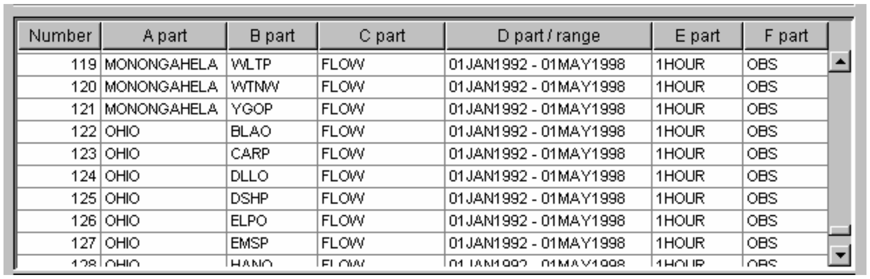
You can also view the pathnames the *Pathname Parts* List (Figure 3.12).



Number	A part	B part	C part	D part	E part	F part
1		AXEMA	FLOW	01OCT2001	1HOUR	OBS
2		AXEMA	PRECIP	01OCT2001	1HOUR	OBS
3		AXEMA	STAGE	01OCT2001	1HOUR	OBS
4		BALD EAGLE TOTAL	FLOW	01OCT2001	1HOUR	N0C0T0
5		BALD EAGLE TOTAL	FLOW	01OCT2001	1HOUR	S0C0T0
6		BALD EAGLE TOTAL	FLOW-CUMLOC	01OCT2001	1HOUR	N0C0T0
7		BALD EAGLE TOTAL	FLOW-CUMLOC	01OCT2001	1HOUR	S0C0T0
8		BALD EAGLE TOTAL	FLOW-HOLDO...	01OCT2001	1HOUR	N0C0T0
9		BALD EAGLE TOTAL	FLOW-HOLDO...	01OCT2001	1HOUR	S0C0T0
10		BALD EAGLE TOTAL	FLOW-UNREG	01OCT2001	1HOUR	N0C0T0

Figure 3.12 HEC-DSSVue – Data Selection List window, pathname parts displayed

The *Condensed Catalog* style (Figure 3.13) abridges time series data sets so that the date span for the entire data set displays in place of the "D" part.



Number	A part	B part	C part	D part / range	E part	F part
119	MONONGAHELA	WLTP	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
120	MONONGAHELA	WTNW	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
121	MONONGAHELA	YGOP	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
122	OHIO	BLAO	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
123	OHIO	CARP	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
124	OHIO	DLLO	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
125	OHIO	DSHP	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
126	OHIO	ELPO	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
127	OHIO	EMSP	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS
128	OHIO	HANO	FLOW	01JAN1992 - 01MAY1998	1HOUR	OBS

Figure 3.13 HEC-DSSVue – Data Selection List window, condensed catalog displayed

3.4.2 Refreshing the Catalog

The catalog automatically refreshes whenever you add, delete or rename records. For very large files, or network files, a dialog box will be shown asking if you want a new catalog. When the catalog is refreshed, it re-inventories all records on disk to get a pathname for every record. To generate a new catalog manually, from the **Edit** menu, click **Refresh Catalog**.

3.4.3 Searching and Filtering HEC-DSS Pathnames

With HEC-DSSVue, you can select specific data sets from a list of pathnames (or *catalog*) in the database. You can refine the list by searching for either a string in the pathnames or for pathname parts.

The following sections describe searching and filtering options.

3.4.4 Searching Pathname Strings

The **Search Pathnames by String** feature allows you to search for records whose pathnames contain specific strings of characters. For example, if you search for “FLOW” the list will display only pathnames containing the string “FLOW” in the pathname.

To **Search Pathnames by String**:

1. From the **View** menu, choose **Search Pathnames by string**.
2. In the **Search Pathname** box (Figure 3.4), type the character string you want to filter the pathnames.
3. Click the **Search** button.

3.4.5 Filtering HEC-DSS Pathnames by Parts

The **Search by Parts** feature lets you choose specific pathname parts from those available in a HEC-DSS file.

To Search by Parts:

- From the **View** menu, click **Search pathnames by parts**.
- Click on the down arrows beside the pathname parts you want to search. This allows you to view lists of available names in the file (Figure 3.14).
- When you have a pathname part list open, you can scroll through the list to locate the pathname part you want.
- Click on a pathname part to select it.
- To return to the previous list, click on the blank field at the top of the list.

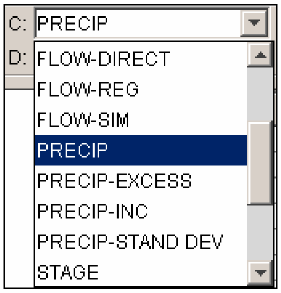


Figure 3.14 Selecting Pathname Parts

3.5 Selecting Pathnames

Once you have filtered a HEC-DSS file for the pathnames you want, you select specific pathnames to visualize data for or perform a utility function on. When you select a Pathname, it will appear in the **Selected Pathnames List** (Figure 3.7).

There are several ways to select pathnames:

- Double-click on an individual pathname in the HEC-DSS Pathname List.
- Highlight a pathname in the HEC-DSS Pathname List then click the **Select** button (Figure 3.1). Until you select a pathname, the **Select** button remains inactive.
- Click and drag your mouse to select a series of pathnames, and then click the **Select** button. You can also use **Ctrl+click** to select multiple, non-consecutive pathnames.
- Hold down the control key and click on pathnames to highlight them, or hold down on the shift key to highlight all pathnames between mouse clicks, and then click the **Select** button.
- If you wish to select all of the pathnames, from the **Edit** menu, choose **Select All**.
- If no pathnames are selected, you can just highlight one or more pathnames and then perform the function that you intended. If pathnames have already been added to the selection box, those pathnames will be used instead of the highlighted ones.

3.5.1 De-Selecting Pathnames

To de-select a pathname currently in the Selected Pathnames List:

1. Click the record's pathname in the Selected Pathnames List to select it.
2. Click the **De-Select** button. The pathname will no longer appear in the Selected Pathnames List.

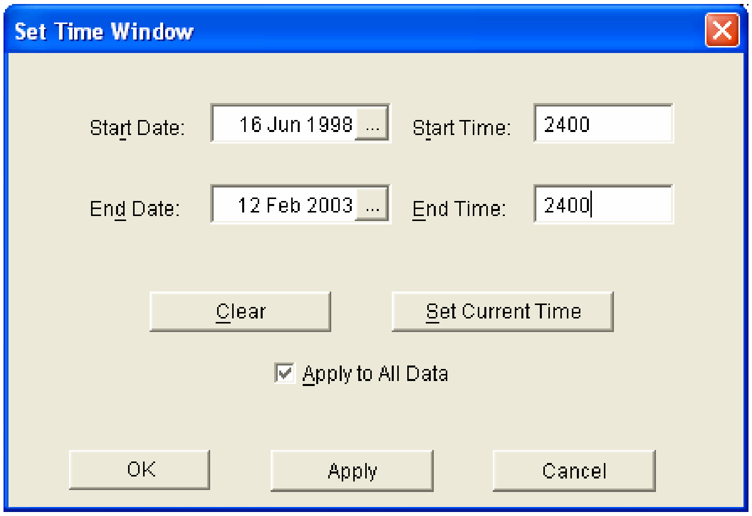
To remove all pathnames from the Selected Pathnames List, click the **Clear Selections** button. To restore selections you have cleared, use the **Restore Selections** button.

3.5.2 Setting the Time Window

You can also specify the time window for data you want displayed in plots and tables. For example, if you have a set of HEC-DSS records ranging from October 1, 1993, to December 30, 1993, you can specify a narrower time window, such as 05OCT1993 to 25OCT1993 for plots and tables you view, and for operations with math functions.

To set the Time Window:

1. Click the **Set Time Window** button. The **Set Time Window** dialog box will open (Figure 3.15).



Start Date: 16 Jun 1998 Start Time: 2400

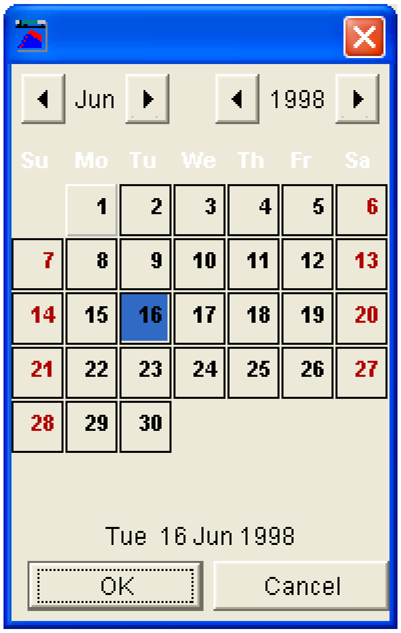
End Date: 12 Feb 2003 End Time: 2400

☑ Apply to All Data

Figure 3.15 Set Time Window Dialog Box

2. You may either enter specific dates and times or use the current time.
   - To use the current time, click the **Set Current Time** button. This will set both the start and end dates and times to the current time.
   - To specify dates and times:
     a. Enter the **Start Date** and **End Date**. Although a variety of formats will be recognized for the date, the typical format is DDMMMYYYY.
     b. Enter the **Start Time** and **End Time** in 24-hour format (e.g., for 2:00 PM, type "1400").
   - You may select the starting and ending dates from the calendar tool, accessible from a button on the right side of the date field. When you click on this button, the selection calendar in a monthly format is opened (Figure 3.16 Calendar Selection Tool). You may scroll through months or years by using the appropriate arrow key at the top of the box.
3. The **Apply to All Data** checkbox is set by default. The time window will be applied to all pathnames that have been selected. To have the time window apply only to pathnames selected after this, uncheck this box.

4. Click the **Apply** button to apply the time window without closing the dialog box.
5. Click **OK** to apply the time window and close the dialog box.



Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Tue 16 Jun 1998

Figure 3.16 Calendar Selection Tool

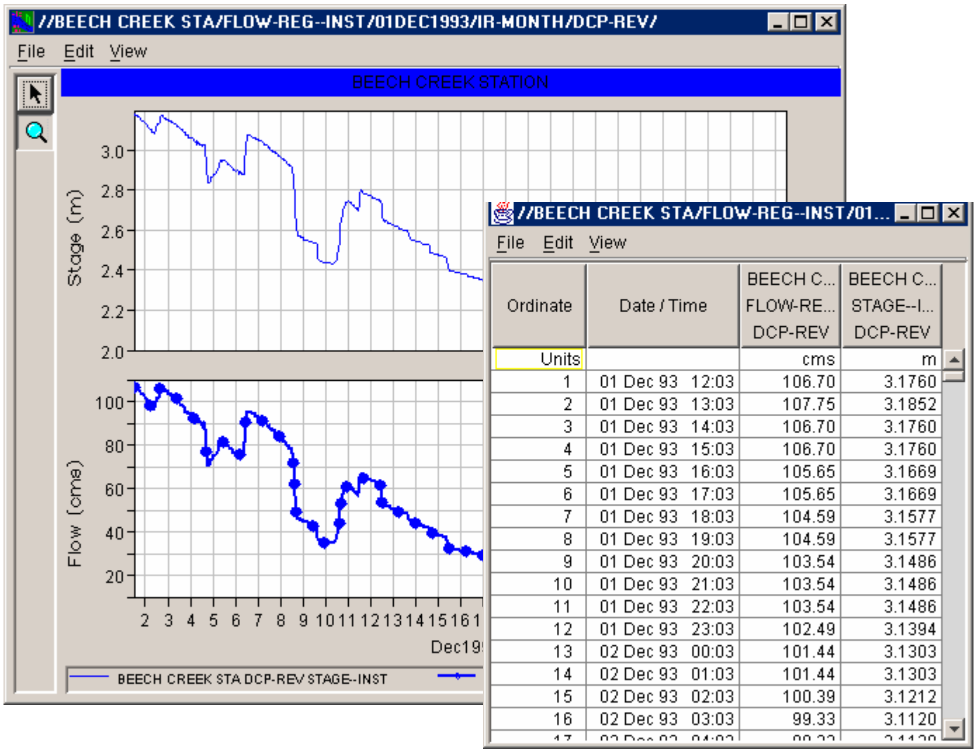
3.6 Visualizing HEC-DSS Data with Plots and Tables

Once the **Selected Pathnames List** contains all the pathnames you wish to display, you can generate plots and tables.

To create a plot or table from the **Display** menu, select **Plot** or **Tabulate**. You can also click on the **Plot** button or the **Tabulate** button.

From a plot window, you can also tabulate data using the **Tabulate** command in the plot's **File** menu. Likewise, to open a plot from a table, you can use the **Plot** command in the table's **File** menu.

Figure 3.17 shows examples of the plots and tables HEC-DSSVue can produce.



Ordinate	Date / Time	BEECH C... FLOW-RE... DCP-REV	BEECH C... STAGE--I... DCP-REV
Units		cms	m
1	01 Dec 93 12:03	106.70	3.1760
2	01 Dec 93 13:03	107.75	3.1852
3	01 Dec 93 14:03	106.70	3.1760
4	01 Dec 93 15:03	106.70	3.1760
5	01 Dec 93 16:03	105.65	3.1669
6	01 Dec 93 17:03	105.65	3.1669
7	01 Dec 93 18:03	104.59	3.1577
8	01 Dec 93 19:03	104.59	3.1577
9	01 Dec 93 20:03	103.54	3.1486
10	01 Dec 93 21:03	103.54	3.1486
11	01 Dec 93 22:03	103.54	3.1486
12	01 Dec 93 23:03	102.49	3.1394
13	02 Dec 93 00:03	101.44	3.1303
14	02 Dec 93 01:03	101.44	3.1303
15	02 Dec 93 02:03	100.39	3.1212
16	02 Dec 93 03:03	99.33	3.1120

Figure 3.17 Example Plot and Table

HEC-DSSVue plots are highly customizable. You can add titles, specify colors and patterns of backgrounds and lines, change line styles, add markers and callouts, add borders to labels, and customize legends, axes, and tic marks.

For more information on plots, refer to the chapter on **Plots**.

For more information about tables, see the chapter on **Utilities**.

3.6.1 Plot Data Options

Two options are provided as a quick way to compare time series data sets for different time periods or for different amounts. The options are turned on or off by selecting them from the submenu and then clicking them.

The **Normalize** option will subtract the differences between the values of first and the second or subsequent data sets from the corresponding data set, so that the curves all start at the same value (which is the value of the first data in the first data set). This is often used to compare cumulative precipitation curves, which usually start at different values.

The **Synch data set times to first** option will subtract the differences between the times of first and the second or subsequent data sets from the corresponding data set, so that the curves all start at the same time (which is the time in the first data set). This is often used to directly compare hydrographs for different years or time periods.

Each option will remain on until you turn it off or the program exits. The options can be used in combination with each other. For example, if you want to compare cumulative precipitation curves for different years, set each option to on. Figure 3.18 shows such an example of normalized cumulative precipitation for years 1986 and 1995, synched to the same start time.

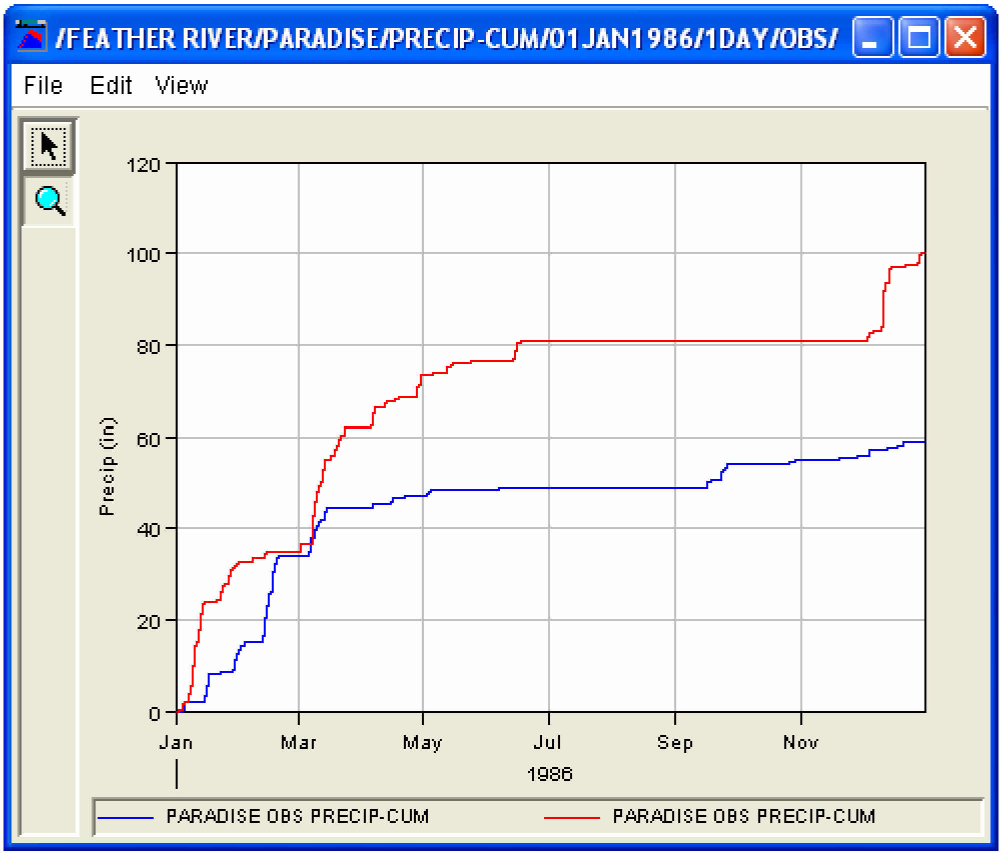


Figure 3.18 Normalized and Time Synched Precipitation Data for Years 1986 and 1995

3.7 Utilities Menu Operations

The **Utilities** menu provides access to **Math Functions**, **Manual Data Entry** (Time Series and Paired Data), and it allows you to **Rename Records**, **Duplicate Records**, **Copy Records**, **Delete Records**, **Undelete Records**, **Merge Files**, **Squeeze**, access the **Script Browser** and the **Script Selector**, and view the HEC-DSS **Status** of files opened.

These functions are discussed in detail in three separate chapters: **Math Functions**, **Utilities** (which also covers tabular editing and printing functions), and **Scripting**.

3.8 Plug-ins

HEC-DSSVue has the capability of accepting Java jar plug-ins. A plug-in is essentially compiled Java code that has been written to extend the capability of HEC-DSSVue. It is similar to a script, but can be more powerful and more customizable. For example, a plug-in might retrieve data, decode it and store it into a HEC-DSS file. Another use might be to perform sophisticated math functions on data selected from the main screen.

Plug-ins are added by placing the jar plug-in file in the HecDssVue/Plugins directory. When HEC-DSSVue is executed, it searches this directory for available plug-ins and loads them into the program. Most plug-ins will add a button to the main toolbar, adjacent to any script buttons. To remove a plug-in, simply remove its jar file from the Plugins directory.

Visit the HEC-DSSVue area of the HEC web site to obtain publicly accessible plug-ins, or contact HEC for more information.

Chapter 4

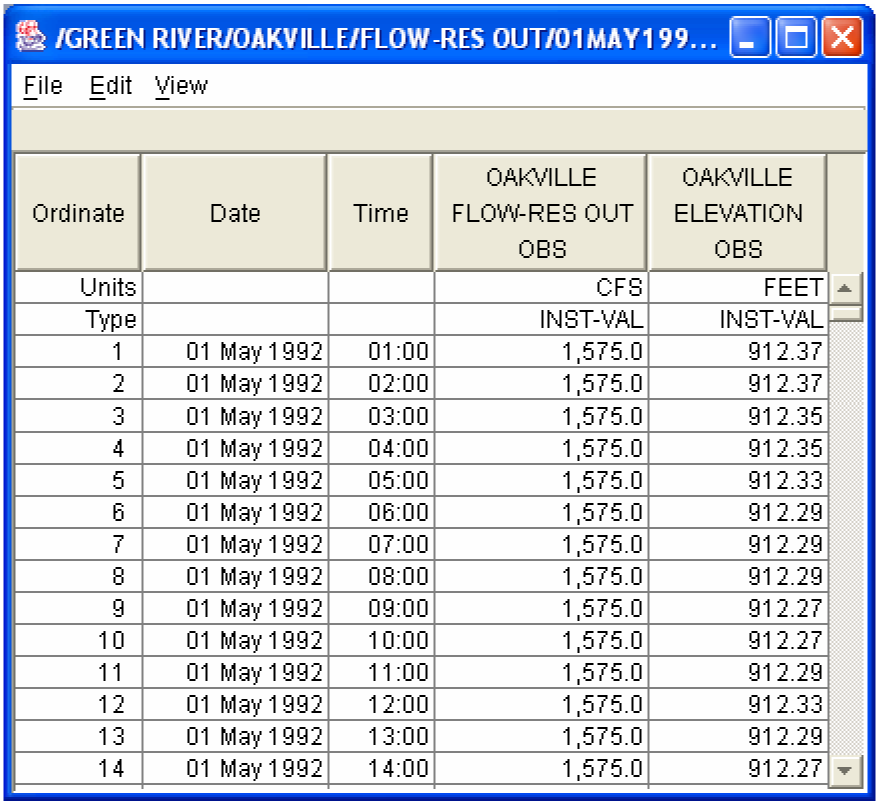
Utilities

HEC-DSSVue utilities allow you to tabulate, edit, duplicate, copy, rename, and delete data, manually enter data, create and edit scripts, and perform math functions, as well as merge and squeeze HEC-DSS files.

This chapter provides guidance in using these utilities and also shows you how to print, copy, and export tabular HEC-DSS data.

4.1 Viewing Tabular Data

Figure 4.1 shows an example table produced using HEC-DSSVue. Tables allow you to view and edit HEC-DSS data in a vertical scrolling window that shows the ordinate (starting from the start date/time), the date and time stamp, and the values for the selected data sets. From the **File** menu of the table window, you can view the tabular data in plot format by clicking **Plot**.



Ordinate	Date	Time	OAKVILLE FLOW-RES OUT OBS	OAKVILLE ELEVATION OBS
Units			CFS	FEET
Type			INST-VAL	INST-VAL
1	01 May 1992	01:00	1,575.0	912.37
2	01 May 1992	02:00	1,575.0	912.37
3	01 May 1992	03:00	1,575.0	912.35
4	01 May 1992	04:00	1,575.0	912.35
5	01 May 1992	05:00	1,575.0	912.33
6	01 May 1992	06:00	1,575.0	912.29
7	01 May 1992	07:00	1,575.0	912.29
8	01 May 1992	08:00	1,575.0	912.29
9	01 May 1992	09:00	1,575.0	912.27
10	01 May 1992	10:00	1,575.0	912.27
11	01 May 1992	11:00	1,575.0	912.29
12	01 May 1992	12:00	1,575.0	912.33
13	01 May 1992	13:00	1,575.0	912.29
14	01 May 1992	14:00	1,575.0	912.27

Figure 4.1 Example Tabulation from HEC-DSSVue

4.1.1 Accessing Tables

To access tables, first select the pathnames of the records you wish to view. There are several ways to select pathnames:

- Double-click on an individual pathname in the HEC-DSS Pathname List.
- Highlight a pathname in the HEC-DSS Pathname List then click the **Select** button. Until you select a pathname, the **Select** button remains inactive.
- Click and drag your mouse to select a series of pathnames, and then click the **Select** button. You can also use **Ctrl+Click** to select multiple, non-consecutive pathnames.
- If you wish to select all of the pathnames for visualization, from the **Edit** menu, click **Select All**.
- If no pathnames are in the selection list, individual pathnames can just be highlighted for quick selection.

When you select a pathname, it will appear in the **Selected Pathnames List**.

Once you have selected the pathnames you want to visualize, you can open a table by clicking on the **Tabulate** ☷ button, or from the **Display** menu, by clicking **Tabulate**.

4.1.2 Customizing the Display of Tabular Data

In HEC-DSSVue tables you have several options for displaying data.

From the **View** menu, you can choose to display commas in numbers by selecting the **Commas** command. The date and time of the data can be tabulated in separate columns by selecting **Date and Time Separately**. To have the dates display the years with four digits, select **Date With 4 Digit Years**. You can set the precision of decimal places for your data by selecting **Decimal Places** and selecting the number of decimal places you wish to display. You can also change to **Show Missing** values as blanks (the default), "-901.0", "M", or "-M-".

4.2 Editing Tabular Data

In HEC-DSSVue, you can edit data directly in tables.

From the **Edit** menu of the table, select **Allow Editing** (Figure 4.2) to manually edit the data in the table. When this option is checked, the background of non-editable table cells is changed to grey, the **Cut** and **Paste** commands become available in the menu, as well as the menu items **Insert Rows**, **Append Rows** and **Delete Rows**.

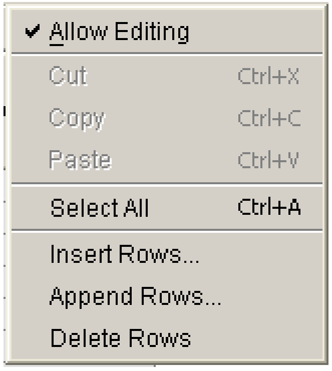


Figure 4.2 Edit menu - Allow Editing option

Currently, you can insert, append or delete rows only when you are working with a single data set.

You can also use **shortcut menu** commands to edit multiple selected cells in tables (Figure 4.3 Shortcut Menu).

If the **Allow Editing** option is disabled (greyed-out), then either you do not have permission to write back to the file, or the table was started from a dialog that does not have access to write to the file (such as a plot).

If you make any edits, HEC-DSSVue prompts you can save your changes with the **Save** or **Save As** items under the **File** menu. If you do not save your data, HEC-DSSVue will prompt you to save your changes when you close the window.

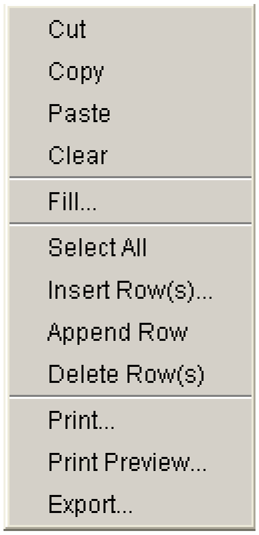


Figure 4.3 Shortcut Menu

4.2.1 Selecting Table Cells

To select an individual table cell, double-click on it.

To select several consecutive cells, click and drag your mouse. You can also use **Ctrl+Click** to select multiple, non-consecutive records.

If you wish to select all rows in a table, from the **shortcut menu**, click **Select All** (Figure 4.3).

4.2.2 Cutting and Pasting Data

The **Cut** command removes data from its current location and places it on the clipboard in ASCII format. For time series data, the **Cut** command will replace the data values with missing flags.

You can cut data from one set of cells and paste them into another set of cells (in the same table or another table). To do this:

1. From the **View** menu of the table (or tables), click **Allow Editing**.
2. Select the cells you want to cut.
3. From either the **View** menu or the **shortcut menu** (Figure 4.3 Shortcut Menu), click **Cut**.
4. Select the cells where you want to move the data.
5. From either the **View** menu or the **shortcut menu** (Figure 4.3 Shortcut Menu), click **Paste**.

4.2.3 Copying and Pasting Data

The **Copy** command places the selected data on the clipboard in ASCII format. You can also copy data from one set of cells to another. To do this:

1. Select **Allow Editing** from the **View** menu of the table (or tables).
2. Select the cells you want to copy.
3. Choose **Copy** from either the **View** menu or the **shortcut menu** (Figure 4.3) of the table.
4. Select the cells where you want to copy the data.
5. Choose **Paste** from either the **View** menu or the **shortcut menu** (Figure 4.3) of the table.

You can also use the **Copy** command to copy and paste data into another application, such as Microsoft Excel or Word.

4.2.4 Clearing Table Cells

When you clear table cells, the data is not saved on the clipboard, so you cannot later paste the data back into the table. For time series data, the **Clear** command will replace the data values with missing flags. To clear table cells:

1. From the **View** menu of the table (or tables), click **Allow Editing**.
2. Select the cells you want to clear.
3. From the **shortcut menu** of the table (Figure 4.3 Shortcut Menu), click **Clear**.

4.2.5 Inserting Rows

For regular-interval time series data, rows can only be inserted at the beginning of the data set. For irregular-interval time series data and paired data, select with your mouse the ordinate of the row where you want rows inserted and then from the **Edit** menu select **Insert Rows…** A small dialog box will be displayed asking you to enter the number of rows to insert. Type in the number of rows you want inserted and press **OK**. That number of blank rows will be inserted at that row moving the original row that you selected down. For irregular-interval time series data, enter the date and time and value for each row, or for paired data, the X and Y ordinate values. Be sure to delete any rows where you have not filled values in.

4.2.6 Appending Rows

Appending rows is automatic when editing data. When you set the table into edit mode, two blank rows appear at the end of the table where you can enter data. After you enter data for a row, an additional blank row appears. When you save the data, any blank rows at the end of the table are removed automatically. You can specify an additional number of rows to append to the data set by selecting the **Append Rows…** option from the **Edit** menu. After you select this, enter the number of rows you want appended to the end of the data set in the dialog box that appears. Delete any rows where you have not filled values in.

4.2.7 Deleting Rows

You may delete rows from a table when working with irregular-interval time series data and paired data. To delete a row from a table, first select the row (see Section 4.2). Next, from the **Edit menu**, click **Delete Rows**.

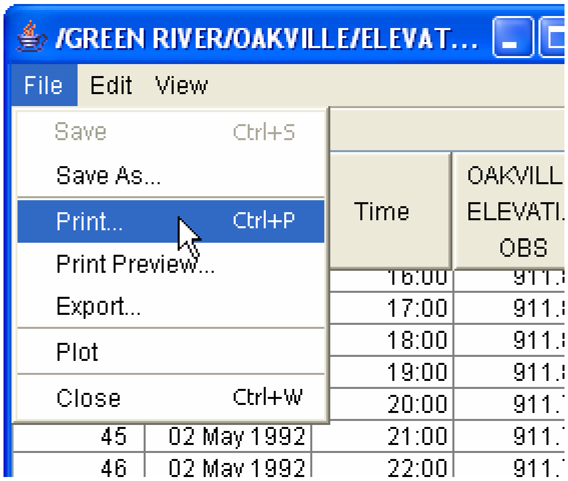
If you are working with regular interval time series data, you will not be able to delete rows; instead, the **Delete Rows** function will change the data values to missing. This is the same as selecting **Clear** from the **shortcut menu**.

4.3 Printing, Copying, and Exporting Tables

HEC-DSSVue tables offer several commands that allow you to print as well as export or copy, and paste tables into other applications such as Microsoft Excel and Word.

4.3.1 Printing Tables

You can access the **Print** and **Print Preview** command from either the **File** menu (Figure 4.4) or from the **shortcut menu** (Figure 4.5) of the table window.



		Time	OAKVILL ELEVATI OBS
		16:00	911.
		17:00	911.
		18:00	911.
		19:00	911.
		20:00	911.
45	02 May 1992	21:00	911.
46	02 May 1992	22:00	911.

Figure 4.4 File Menu--Table Window

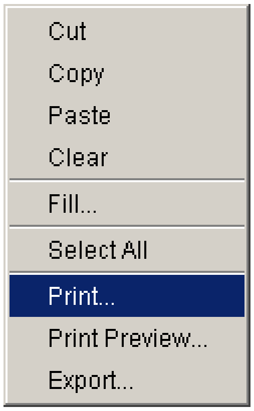


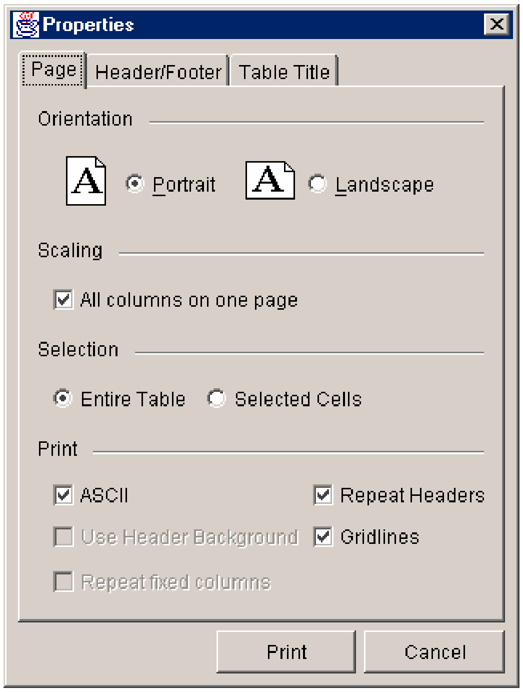
Figure 4.5 Shortcut Menu--Print Command

In a table, the **Print** and **Print Preview** commands open the **Print Properties** dialog box (Figure 4.6), which offers options on three tabs.

The **Page** tab allows you to specify the page Orientation, Scaling, and Selection; you can also choose to print the table as ASCII, Repeat Headers on every page, and print the Gridlines.

On the **Header/Footer** tab, you can type in the header and footer you want to appear on your printed pages.

The **Table Title** tab offers a default title for the table based on the data source. You may edit this title.



Orientation

● Portrait ○ Landscape

Scaling

☑ All columns on one page

Selection

● Entire Table ○ Selected Cells

Print

☑ ASCII ☑ Repeat Headers

☐ Use Header Background ☑ Gridlines

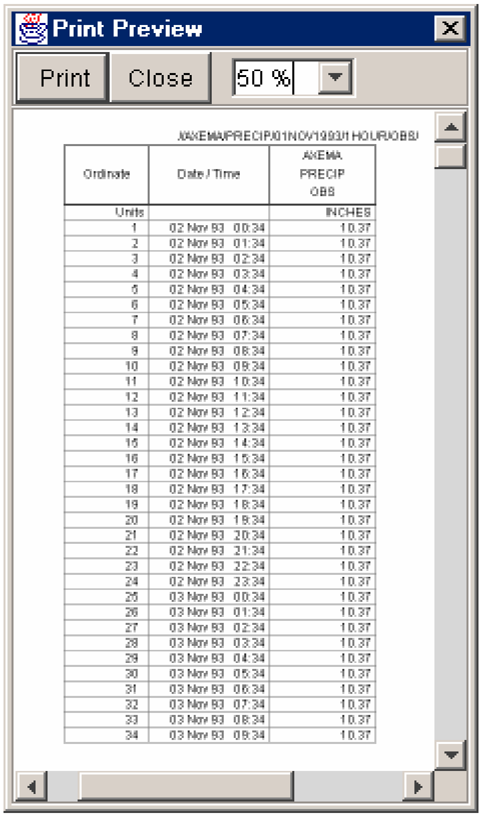
☐ Repeat fixed columns

Figure 4.6 Print Properties Dialog Box

On the **Print Properties** dialog box, the **Print** button performs two functions, depending on whether you arrived at the dialog box via the **Print** command or the **Print Preview** command.

From the **Print Preview** command on the shortcut menu, the **Print** button on the **Print Properties** dialog box opens a **Print Preview** window, which allows you to view the table as it will be printed. Figure 4.7 shows an example. You can click the **Print** button in the Print Preview window to print your table.

From the **Print** command, the **Print** button on the **Print Properties** dialog box opens a Windows-style print dialog box, where you can choose your printer, set printer properties, and specify the number of copies to print. You can also print your table to a file instead of to a printer.



/AXEMA/PRECIP/01NOV1993/1HOUR/OBS/

Ordinate	Date / Time	AXEMA PRECIP OBS
Units		INCHES
1	02 Nov 93 00:34	10.37
2	02 Nov 93 01:34	10.37
3	02 Nov 93 02:34	10.37
4	02 Nov 93 03:34	10.37
5	02 Nov 93 04:34	10.37
6	02 Nov 93 05:34	10.37
7	02 Nov 93 06:34	10.37
8	02 Nov 93 07:34	10.37
9	02 Nov 93 08:34	10.37
10	02 Nov 93 09:34	10.37
11	02 Nov 93 10:34	10.37
12	02 Nov 93 11:34	10.37
13	02 Nov 93 12:34	10.37
14	02 Nov 93 13:34	10.37
15	02 Nov 93 14:34	10.37
16	02 Nov 93 15:34	10.37
17	02 Nov 93 16:34	10.37
18	02 Nov 93 17:34	10.37
19	02 Nov 93 18:34	10.37
20	02 Nov 93 19:34	10.37
21	02 Nov 93 20:34	10.37
22	02 Nov 93 21:34	10.37
23	02 Nov 93 22:34	10.37
24	02 Nov 93 23:34	10.37
25	03 Nov 93 00:34	10.37
26	03 Nov 93 01:34	10.37
27	03 Nov 93 02:34	10.37
28	03 Nov 93 03:34	10.37
29	03 Nov 93 04:34	10.37
30	03 Nov 93 05:34	10.37
31	03 Nov 93 06:34	10.37
32	03 Nov 93 07:34	10.37
33	03 Nov 93 08:34	10.37
34	03 Nov 93 09:34	10.37

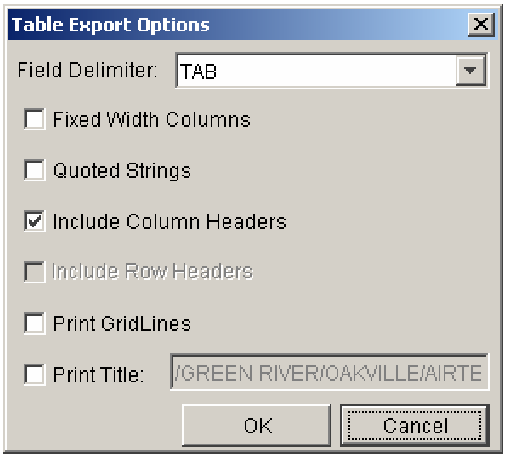
Figure 4.7 Print Preview of a Table

4.3.2 Exporting Tables

From the **shortcut menu**, click **Export** to export a table to a file, which you can then open in another application.

The **Export** command opens the **Table Export Options** dialog box (Figure 4.8).

In the **Table Export Options** dialog box, you can choose the **Field Delimiter** (tab, space, comma, or colon), specify **Fixed-Width Columns**, choose to display **Quoted Strings**, **Include Column Headers**, and opt to **Print Gridlines** and **Title**.



Field Delimiter: TAB

☐ Fixed Width Columns

☐ Quoted Strings

☑ Include Column Headers

☐ Include Row Headers

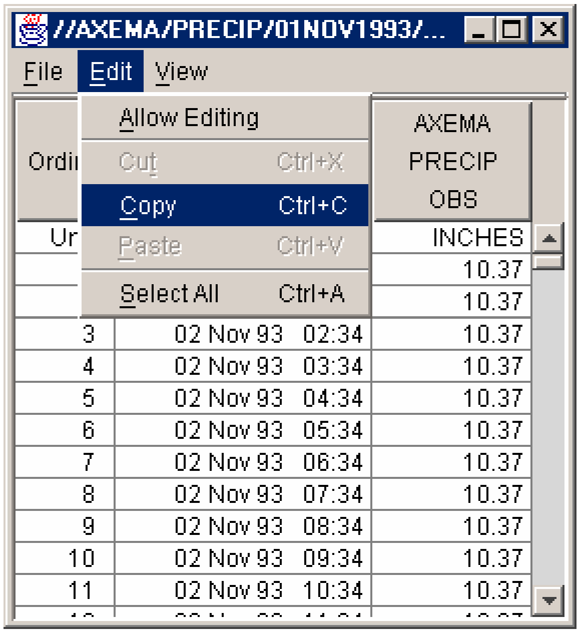
☐ Print GridLines

☐ Print Title: /GREEN RIVER/OAKVILLE/AIRTE

Figure 4.8 Table Export Options Dialog Box

4.3.3 Copying Tables to the Clipboard for Use in Other Applications

To copy a table to the clipboard, click **Copy** from the table window's **Edit** menu (Figure 4.9). You can also right-click inside the table and select **Copy** from the **shortcut menu** (Figure 4.10). You can then paste the table as tab-separated values into another application such as Microsoft Excel or Word.



Ordi		AXEMA PRECIP OBS
Ur		INCHES
		10.37
		10.37
3	02 Nov 93 02:34	10.37
4	02 Nov 93 03:34	10.37
5	02 Nov 93 04:34	10.37
6	02 Nov 93 05:34	10.37
7	02 Nov 93 06:34	10.37
8	02 Nov 93 07:34	10.37
9	02 Nov 93 08:34	10.37
10	02 Nov 93 09:34	10.37
11	02 Nov 93 10:34	10.37

Figure 4.9 Edit Menu--Table Window

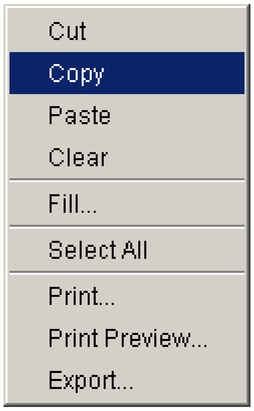


Figure 4.10 Shortcut Menu--Copy Command

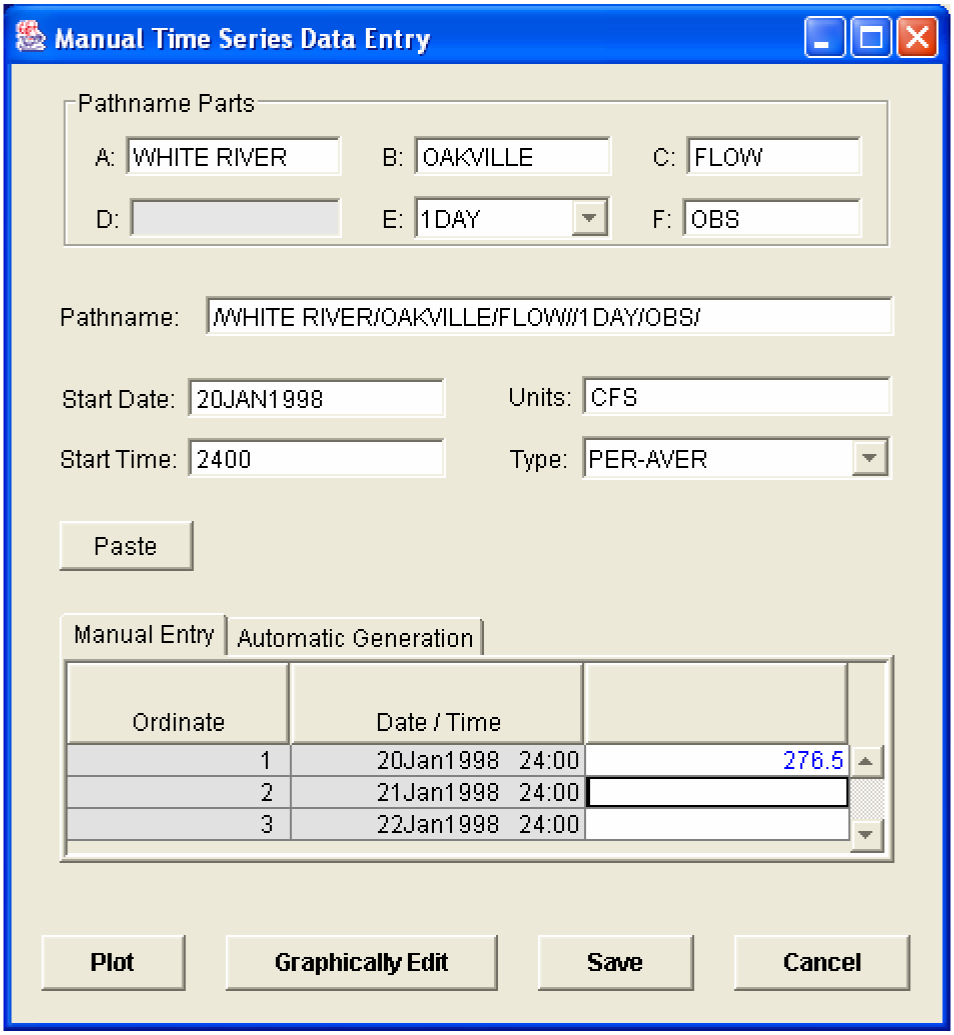
4.4 Entering Data Manually

You can enter time series and paired data manually using the Manual Data Entry editors available from the **Utilities** menu.

4.4.1 Entering Time Series Data Manually

To enter time series data manually:

1. From the **Utilities** menu, choose **Manual Data Entry**, and then select **Time Series**. The **Manual Time Series Data Entry** dialog box (Figure 4.11) will open.



Manual Time Series Data Entry

Pathname Parts

A: WHITE RIVER B: OAKVILLE C: FLOW

D: E: 1DAY F: OBS

Pathname: /WHITE RIVER/OAKVILLE/FLOW//1DAY/OBS/

Start Date: 20JAN1998 Units: CFS

Start Time: 2400 Type: PER-AVER

Ordinate	Date / Time	
1	20Jan1998 24:00	276.5
2	21Jan1998 24:00	
3	22Jan1998 24:00	

Figure 4.11 HEC-DSSVue Manual Time Series Data Entry Dialog Box

2. Type the **Pathname Parts** into the A, B, C, and F boxes, then select the appropriate time interval for the E box. The complete pathname will automatically appear in the **Pathname** box. You can also enter the pathname into the **Pathname** box; the parts will appear in the **Pathname Parts** boxes. You cannot enter the "D" (date) part, as this is set according to your **Start Date**).
3. Enter the **Start Date** (*e.g.,* 25Mar2002) and **Start Time** (*e.g.,* 1400).
4. Enter the **Units** (*e.g.,* CFS)
5. From the **Type** list, select a data type. Your options are *PER-AVER*, *INST-VAL*, *PER-CUM*, and *INST-CUM*.
6. For regular-interval time series data, the Date/Time boxes in the table will fill in automatically according to the start date and time you have entered. For irregular-interval data, you will need to enter a date and time for each data value.
7. Type the data values into the third column.
8. To view the data in plot form, click the **Plot** button.
9. To graphically edit the data you have entered, click **Graphically Edit**.
10. To save the new time series record, click **Save**.

You can paste data on the clipboard from another application by clicking the **Paste** button. This allows you to copy data from various applications like Microsoft Excel into HEC-DSS. When you paste data into HEC-DSS, the tab characters and carriage return characters in the program that you are copying from must match the data entry table. Tab characters move to the next column to the right, while carriage return characters move to the next row down. These are the default characters used in Excel.

For regular interval time series data, the data must be in a columnar format (e.g., a column from Microsoft Excel) and should contain data values only. For irregular interval data, the date and time must precede each data value, with tab characters separating them. For paired data, all values for each ordinate must be separated by a tab character. To enter data using the paste capability:

1. Complete steps 1 through 5 (above.)
2. Select your data set from the other application and **Copy** it into the clipboard.
3. Press the **Paste** button located just above the table. You should see the data from the clipboard appear in the table.

You can also automatically generate regular-interval time series data. This will fill in a single number for a time window. To generate this data:

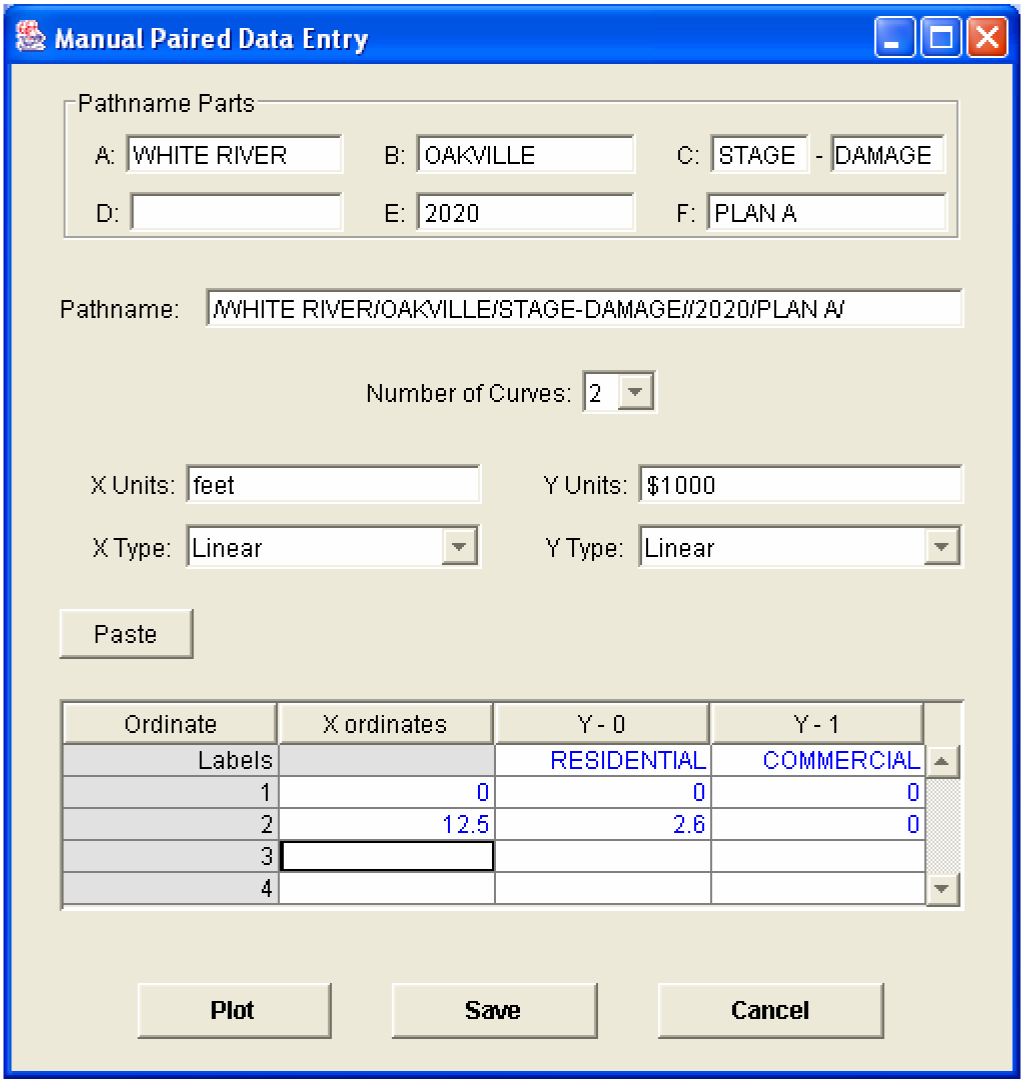
1. Complete steps 1 through 5 (above.)

2. Select the **Automatic Generation** tab.
3. Enter the **End Date** and **End Time** for the data.
4. Enter the **Fill Value**, which is the single value for the specified time window.
5. Press the **Generate** button. This will return you to the **Manual Entry** tab, where you can plot, save, or further edit the data.

4.4.2 Entering Paired Data Manually

To enter paired data manually:

1. From the **Utilities** menu, choose **Manual Data Entry**, and then select **Paired Data**. The **Manual Paired Data Entry** dialog box dialog box (Figure 4.12) will open.



Manual Paired Data Entry

Pathname Parts

A: WHITE RIVER B: OAKVILLE C: STAGE - DAMAGE

D: E: 2020 F: PLAN A

Pathname: /WHITE RIVER/OAKVILLE/STAGE-DAMAGE//2020/PLAN A/

Number of Curves: 2

X Units: feet Y Units: $1000

X Type: Linear Y Type: Linear

Ordinate	X ordinates	Y - 0	Y - 1
Labels		RESIDENTIAL	COMMERCIAL
1	0	0	0
2	12.5	2.6	0
3			
4			

Figure 4.12 HEC-DSSVue Manual Paired Data Entry Dialog Box

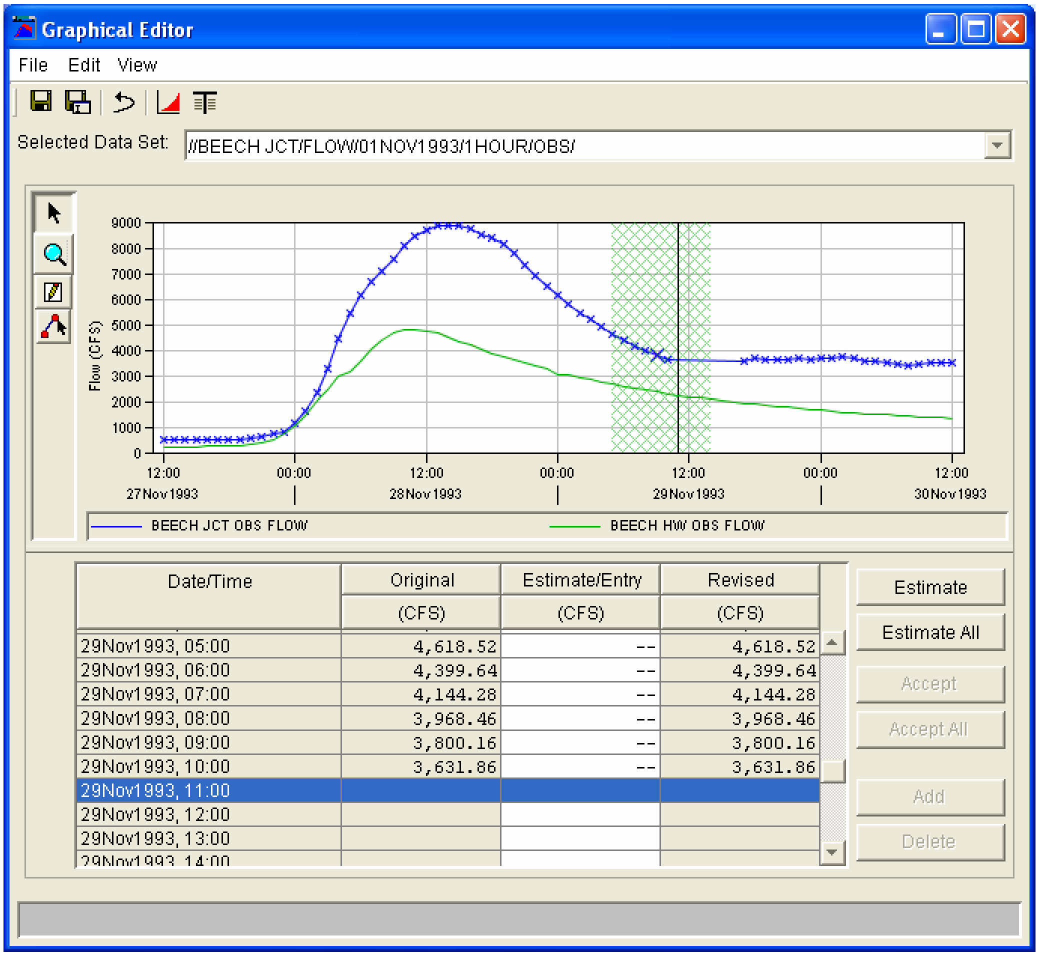
1. Type the **Pathname Parts** into the A, B, C, D, E, and F boxes. The complete pathname will automatically appear in the **Pathname** box. You can also enter the pathname into the **Pathname** box; the parts will appear in the **Pathname Parts** boxes. Be sure the "C" part contains both an X parameter and a Y parameter, separated by a hyphen (e.g., STAGE-FLOW or ELEV-DAMAGE).
2. Select the **Number of Curves** for the Y parameter from the list.
3. Enter the **X Units** (belonging to the first parameter) and the **Y Units** (belonging to the second parameter).
4. Choose the **X Type** and **Y Type** from the lists. Available options are *Linear*, *Log*, and *Probability.*
5. In the table, the **Y ordinates** column will split into individual columns according to the **Number of Curves** you have specified.
6. Type the data values into the **X ordinates** and **Y ordinates** columns.
7. To view the data in plot form, click the **Plot** button.
8. To save the new time series record, click **Save**.

You can paste data on the clipboard from another application by clicking the **Paste** button. This allows you to copy data from various applications like Microsoft Excel into HEC-DSS. When you paste data into HEC-DSS, the tab characters and carriage return characters in the program that you are copying from must match the data entry table. Tab characters move to the next column to the right, while carriage return characters move to the next row down. These are the default characters used in Excel. For paired data, all values for each ordinate must be separated by a tab character with a carriage return character at the end of each row. To enter data using the paste capability:

1. Complete steps 1 through 6 (above.)
2. Select your data set from the other application and **Copy** it into the clipboard.
3. Press the **Paste** button located just above the table. You should see the data from the clipboard appear in the table.

4.5 Graphical Editing

You can view and edit time series data in a combined plot/table using the **Graphical Editor** for HEC-DSSVue. An example of the Graphical Editor is shown in Figure 4.13. To access the HEC-DSSVue **Graphical Editor**, select one or more pathnames from the main HEC-DSSVue screen and then select **Graphical Edit…** from the **Edit** menu, or click on the Graphical Edit button. Only time series data can be edited through the graphical editor.



Selected Data Set: //BEECH JCT/FLOW/01NOV1993/1HOUR/OBS/

Date/Time	Original	Estimate/Entry	Revised
	(CFS)	(CFS)	(CFS)
29Nov1993, 05:00	4,618.52	--	4,618.52
29Nov1993, 06:00	4,399.64	--	4,399.64
29Nov1993, 07:00	4,144.28	--	4,144.28
29Nov1993, 08:00	3,968.46	--	3,968.46
29Nov1993, 09:00	3,800.16	--	3,800.16
29Nov1993, 10:00	3,631.86	--	3,631.86
29Nov1993, 11:00			
29Nov1993, 12:00			
29Nov1993, 13:00			
29Nov1993, 14:00			

Figure 4.13 HEC-DSSVue Graphical Editor

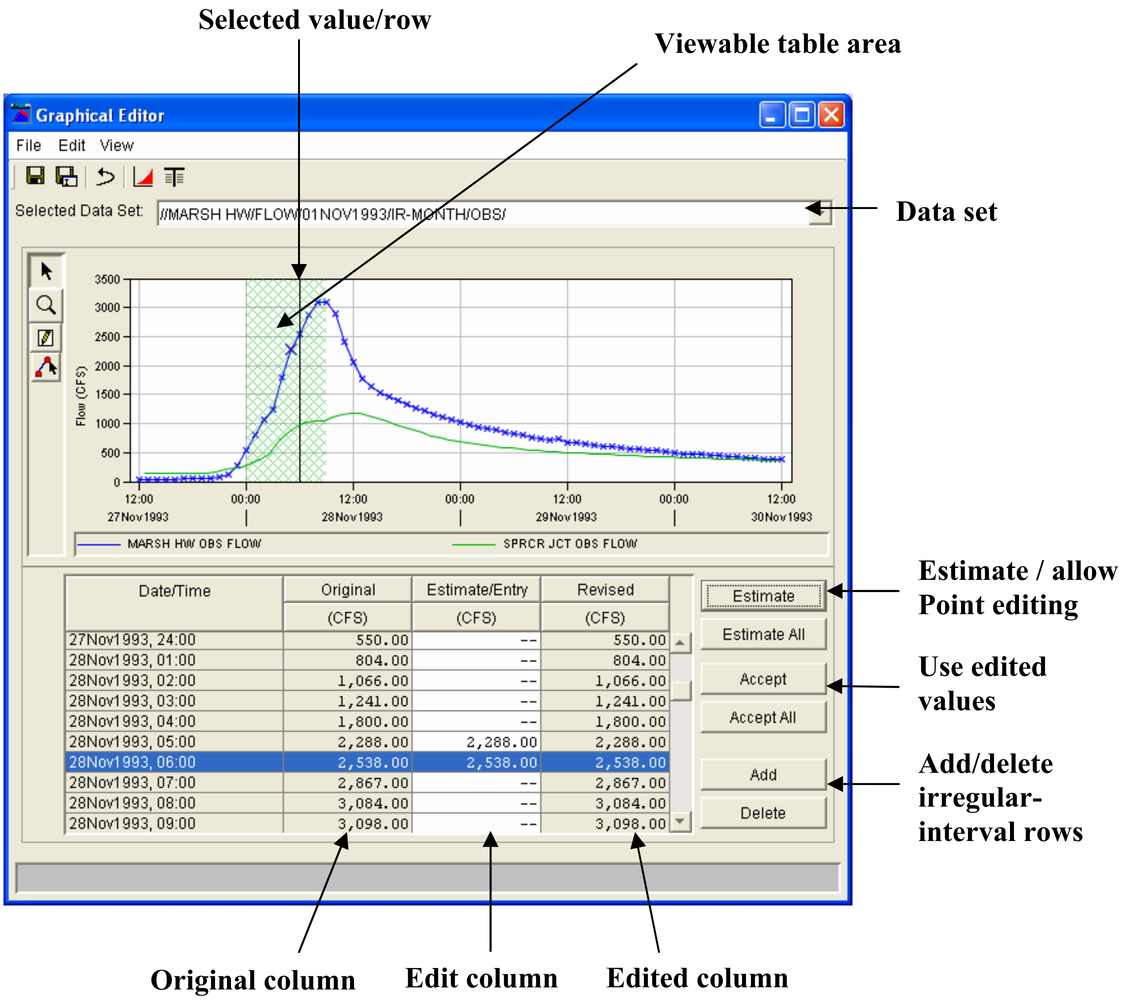
The Graphical Editor displays an editable plot of the data on the top of the window and a scrollable table of the data on the bottom of the window. The area of the data that is visible in the table is marked with a green hash region in the plot. As you scroll through the table, the green hash area moves along with the data that is displayed. When you select a point on the curve or in the table, a black vertical line is drawn at that point to show which value has been selected.

You can edit data in the graphical editor by drawing a curve with your mouse, or by selecting a point on the curve and moving that point up or down, or by changing a value in the associated table. You can also have the editor linear interpolate missing points, or use the other fill options associated with tables.

You can only edit one curve at a time, but you can plot other data sets on the same graph for comparison purposes. You can also select multiple data sets and scroll through them as you edit.

4.5.1 Graphical Editor Components

Figure 4.14 shows the major components of the graphical editor.



Selected Data Set: //MARSH HW/FLOW/01NOV1993/IR-MONTH/OBS/

Date/Time	Original	Estimate/Entry	Revised
	(CFS)	(CFS)	(CFS)
27Nov1993, 24:00	550.00	--	550.00
28Nov1993, 01:00	804.00	--	804.00
28Nov1993, 02:00	1,066.00	--	1,066.00
28Nov1993, 03:00	1,241.00	--	1,241.00
28Nov1993, 04:00	1,800.00	--	1,800.00
28Nov1993, 05:00	2,288.00	2,288.00	2,288.00
28Nov1993, 06:00	2,538.00	2,538.00	2,538.00
28Nov1993, 07:00	2,867.00	--	2,867.00
28Nov1993, 08:00	3,084.00	--	3,084.00
28Nov1993, 09:00	3,098.00	--	3,098.00

Figure 4.14 HEC-DSSVue Graphical Editor Components

The components of the graphical editor are:

- **Selected value/row** shows what data value is selected on the plot and its corresponding row in the table.
- **Viewable table area** indicates the area in the plot that corresponds to the data showing in the table.
- **Data Set** indicates which data set is being edited.
- **Estimate** generates a linear interpolation for selected rows where there is missing data or copies the original data into the editable field. **Estimate All** performs this for all rows in the table.
- **Accept** copies data from the Estimate/Entry (edit) column into the edited column for the selected rows. This is for intermediate saving of the last data edited. **Accept All** copies all data from the edit column into the edited column. You select this if you want to retain all changed values.
- **Add** adds rows for irregular-interval time series data. A dialog will be displayed to set the date and time for the row added.
- **Delete** removes rows for irregular-interval time series data. You cannot remove regular-interval data; you can only change it to missing.
- **Original column** contains the values of the original data set. The values are generally depicted with an orange curve in the plot with each point marked by an "x". If the value is the same as the one in the **Revised** column, that point will be over drawn with the blue revised value.
- **Estimate/Entry (Edit) column** is the column where data can be manually entered or edited with the point tool. These values are marked with magenta triangles in the plot. When you are satisfied with the changes, press the **Accept** or **Accept All** button and the values will display in the Revised column.
- **Revised (Edited) column** contains the final values that will be saved when **Save** or **Save As** is selected. These values are colored blue and marked with an "x".

4.5.2 Graphical Editor Tools and Buttons

Figure 4.15 shows the mouse tools and toolbar buttons available in the graphical editor.

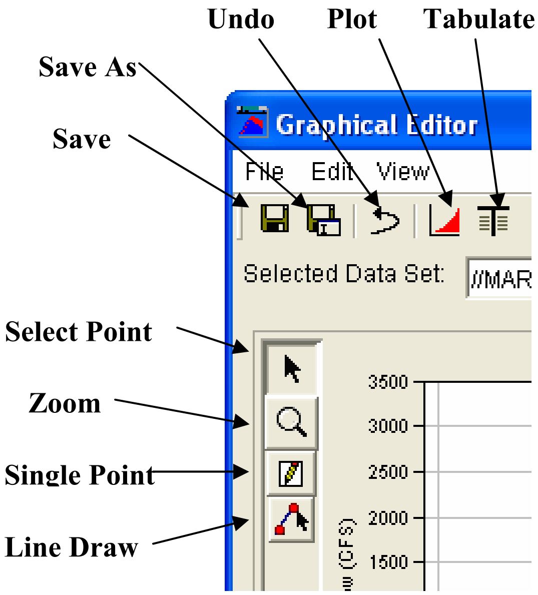


Figure 4.15 Graphical Edit tools

The mouse tools are:

- **Select Point tool** selects a point on the curve to edit by moving the mouse to that point and pressing the left mouse button.
- **Zoom tool** put the mouse in zoom mode. To zoom-in, move the mouse to one corner of the area you want to zoom-in to, press and hold down the left mouse button while dragging the mouse to the opposite corner and the release the button. To zoom-out, press the right mouse button. The graph is zoomed-out in increments.
- **Single Point Edit tool** puts the mouse in a point edit mode, as described in Section 4.5.4.
- **Line Draw (multi-point edit) tool** puts the mouse in a line draw mode, as described in Section 4.5.3.

The toolbar buttons are:

- **Save button** saves the data in the Revised column to the DSS file without changing the pathname.

- **Save As button** saves the data in the Revised column to the DSS file after allowing you to change one or more of the pathname parts.
- **Undo button** restores the values to the original data set, undoing **all** changes.
- **Plot button** plots the original and revised data in a standard plot window
- **Tabulate button** tabulates the original and revised data in a standard table.

4.5.3 Editing a Curve with the Line Draw Tool

You can edit data by drawing a curve freehand (actually mouse-hand) by moving the mouse from left to right while clicking on the left mouse button at selected inflection points. To do this:

1. Select the data set from the drop down box at the top of the screen if it is not selected already.
2. Press the **Line Draw (multi-point edit)** tool.
3. Move the mouse to the left-most position of the curve where you want to start editing.
4. Click on the left mouse button to mark the first data point.
5. Move the mouse right along your desired curve and press the left mouse button at inflection points. The graphical editor will draw a straight line between points.
6. If you make a mistake, you can move the mouse left past that point and insert a new inflection point. You can backup as many points at one time as you want.
7. When you are finished with a curve segment, press the right mouse button. (That location will not be used as part of the curve.) The line segments just given will be drawn with magenta triangles, and those data points entered in the **Estimate/Entry** column.
8. You can draw other curve segments elsewhere on the plot using the same procedure.
9. If you need to remove a curve segment that you have drawn, select the values for it in the table and press the **Estimate** button. The original values will be copied into the **Estimate/Entry** column.
10. After completing your curve, press the **Accept** button or the **Accept All** button to accept all edits. This will copy the data from the **Estimate/Entry** column into the **Revised** column and will be ready to save to the HEC-DSS file.

11. At the end of all of your edits, press the **Save** or **Save As** button or select that item from the **File** menu to store your changes in the HEC-DSS file.

4.5.4 Editing a Point with the Single-Point Edit Tool

You can edit single data values by double clicking on the new point that you want or clicking on the existing point and moving it vertically. To do this:

1. Select the data set from the drop down box at the top of the screen if it is not selected already.
2. Press the **Single-Point Edit** tool.
3. Find the new value of the point on the graph that you want to edit and double click the mouse at that location. This will enter the value that you selected into the **Estimate/Entry** column.
4. Edit the remaining points that you desire to have changed and when you are complete, press the **Accept** button or the **Accept All** button to accept all edits. This will copy the data from the **Estimate/Entry** column into the **Revised** column and will be ready to save to the HEC-DSS file.
5. At the end of all of your edits, press the **Save** or **Save As** button or select that item from the **File** menu to store your changes in the HEC-DSS file.

You can also grab the point that you want to modify and move it vertically. To do this:

1. Click the left mouse button on the data point you want to edit. The table row containing that point will be highlighted.
2. Press the **Estimate** button to provide a value to start from, unless there is already a value in the Estimate column. A magenta triangle will be drawn at that point on the curve.
3. Move the mouse over the triangle and press and hold down the left mouse button. Move the mouse up or down to the desired value.
4. Edit the remaining points that you desire to have changed and when you are complete, press the **Accept** button or the **Accept All** button to accept all edits. This will copy the data from the **Estimate/Entry** column into the **Revised** column and will be ready to save to the HEC-DSS file.
5. At the end of all of your edits, press the **Save** or **Save As** button or select that item from the **File** menu to store your changes in the HEC-DSS file.

4.5.5 Editing Data in the Table.

You can edit data in the table, and add or delete irregular-interval time series values, although you cannot extend the table. Regular-interval values cannot be deleted or added, however you can change values to missing. To edit data within the table:

1. Select the data set from the drop down box at the top of the screen if it is not selected already.
2. Select the row in the table that you want to edit at, or select the location on the graph with the mouse after pressing the **Select** tool.
3. Type in the new values in the **Estimate/Entry** column. To move to the next row in the column, press the Tab key.
4. You can copy the original value to the **Estimate/Entry** column and then change it by pressing the **Estimate** or **Estimate All** button.
5. When complete, press the **Accept** button or the **Accept All** button to accept all edits. This will copy the data from the **Estimate/Entry** column into the **Revised** column and will be ready to save to the HEC-DSS file.
6. At the end of all of your edits, press the **Save** or **Save As** button or select that item from the **File** menu to store your changes in the HEC-DSS file.

You can repeat values or linearly interpolate between selected values or delete values by using the popup selector shown in Figure 4.16.

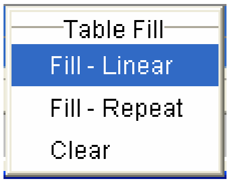


Table Fill
Fill - Linear
Fill - Repeat
Clear

Figure 4.16 Table Fill Options Box

To fill multiple values in the table:

1. Select (highlight) the starting row of the data that you want to fill. For linear and repeat fill, this marks the starting value to use, which will not be changed. If the clear fill option is selected, this is the first value to be cleared.
2. Mark the ending row. You can do this by holding down the left mouse button and moving the mouse down the table, or by left-clicking the mouse on the ending row while holding down the "Shift" key.
3. Right click the mouse button to bring up the popup window in Figure 4.16 and then select either **Fill – Linear**, **Fill – Repeat**, or **Clear**. For **Linear**, the values between the first and last point will be linearly interpolated,

with the first and last point remaining the same. For **Repeat**, the first value will be repeated through the last value. For **Clear**, all values within the selected rows will be set to missing.

4. When complete, press the **Accept** button or the **Accept All** button to accept all edits. This will copy the data from the **Estimate/Entry** column into the **Revised** column and will be ready to save to the HEC-DSS file.

5. At the end of all of your edits, press the **Save** or **Save As** button or select that item from the **File** menu to store your changes in the HEC-DSS file.

4.5.6 Printing the Table and Graph

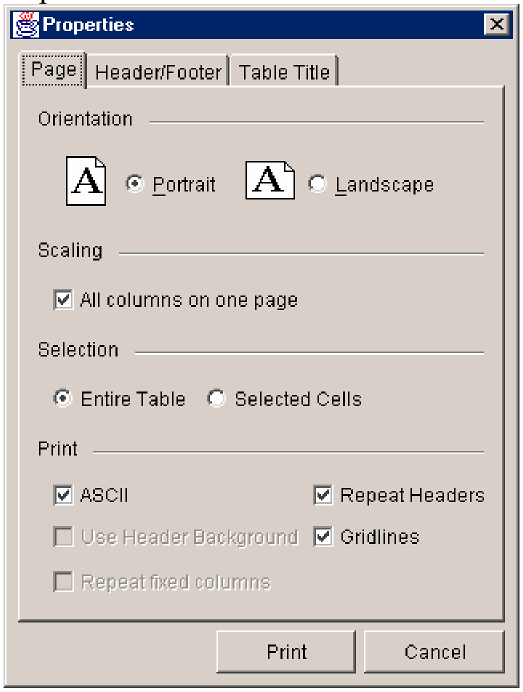
You can print the table or graph by using the **Print Table** or **Print Graph** command from the **File** menu of the Graphical Editor window.

The **Print Table** commands open the **Print Properties** dialog box (Figure 4.17), which offers options on three tabs.

The **Page** tab allows you to specify the page Orientation, Scaling, and Selection; you can also choose to print the table as ASCII, Repeat Headers on every page, and print the Gridlines.

On the **Header/Footer** tab, you can type in the header and footer you want to appear on your printed pages.

The **Table Title** tab offers a default title for the table based on the data source. You may edit this title.



Orientation

● Portrait ○ Landscape

Scaling

☑ All columns on one page

Selection

● Entire Table ○ Selected Cells

Print

☑ ASCII ☑ Repeat Headers

☐ Use Header Background ☑ Gridlines

☐ Repeat fixed columns

Figure 4.17 Print Table Properties Dialog Box

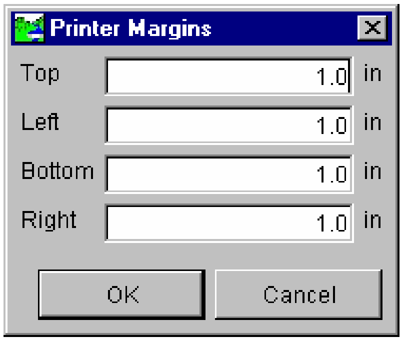
When you press the **Print** button, a standard Windows-style print dialog box opens to select the printer and printer options.

The **Print Graph** command opens a standard Windows-style print dialog box to select the printer and printer options for printing the graph. The printed graph will include the shaded hash area.

You can set the page settings for the printed graph. To do this, click **Page Setup** from the **File** menu.

From the **File** menu, click **Page Setup**. The **Page Setup** dialog box will open (Figure 4.18). Here you can set the page **Orientation**, **Page Margins, Page Numbers**, and **Printer Scale**.

Set Margins opens the **Printer Margins** dialog box (Figure 4.19).



Printer Margins

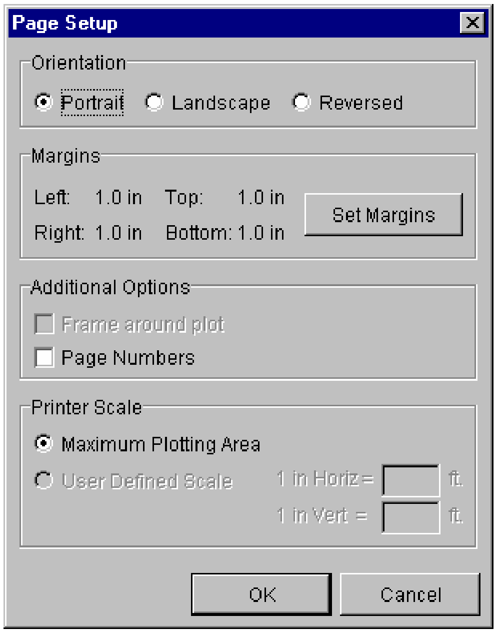
Top 1.0 in

Left 1.0 in

Bottom 1.0 in

Right 1.0 in

Figure 4.19 Printer Margins Dialog Box



Page Setup

Orientation

● Portrait ○ Landscape ○ Reversed

Margins

Left: 1.0 in Top: 1.0 in

Right: 1.0 in Bottom: 1.0 in

Additional Options

☐ Frame around plot

☐ Page Numbers

Printer Scale

● Maximum Plotting Area

○ User Defined Scale 1 in Horiz = ft.

1 in Vert = ft.

Figure 4.18 Page Setup Dialog Box

4.5.7 Graphical Editor View Options

From the **View** menu, you can choose to plot the other data sets that you have selected on the same plot at the same time by selecting the **Show Comparison Data** item. If the data types are the same, the data sets will plot in the same viewport; otherwise they will be plotted in different viewports.

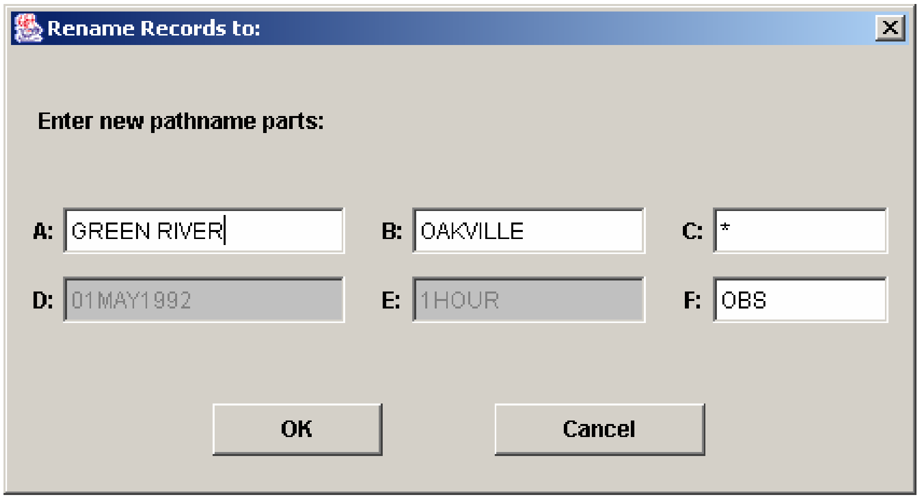
You cannot edit the other data sets at the same time. To edit comparison data sets, you must select a data set from the **Selected Data Set** box under the menu area.

You can also display a standard table of the selected data (original and modified) from the **Tabulate** item, or a standard plot from the **Show Plot** item. You can set the precision of decimal places for displaying your data by selecting **Decimal Places** and selecting the number of decimal places you wish to display.

4.6 Renaming HEC-DSS Data in HEC-DSSVue

The **Rename** command renames record pathnames. To rename HEC-DSS records:

1. Select the record or records to rename.
2. From the **Utilities** menu, select **Rename Records**. The **Rename Records to:** dialog box (Figure 4.20) will open.
3. You may change one or more pathname parts or the entire record. Type the new **Pathname Parts** into the A, B, C, D, E, and F boxes. You cannot change the D or E parts for time series data (use Math Functions to accomplish this). For multiple records, pathname parts that are the same for all records will show up in the dialog. Where parts differ (such as having pathnames with C parts of FLOW and ELEVATION), the part will be displayed with an asterisk (*).
4. Click **OK**.



Enter new pathname parts:

A: GREEN RIVER
B: OAKVILLE
C: *
D: 01MAY1992
E: 1HOUR
F: OBS

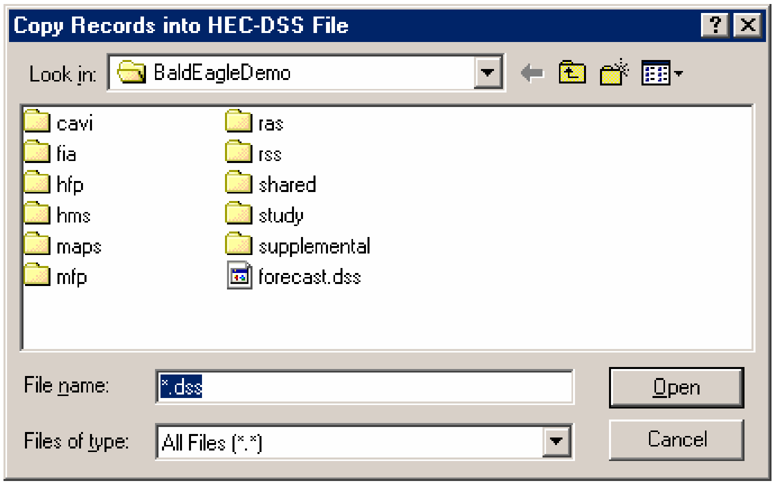
Figure 4.20 HEC-DSSVue Rename Records to Dialog Box

4.7 Copying Records into another HEC-DSS File

To copy records into another HEC-DSS file,

1. Select the record or records to copy.
2. From the **Utilities** menu, select **Copy Records**. The **Copy Records into HEC-DSS File** dialog box (Figure 4.21) will open.
3. In the **File Name** box, type in a new HEC-DSS filename or select an existing HEC-DSS File into which you want to copy the record into and click **Open**. A confirmation message will appear stating that the record has been copied to the HEC-DSS file you selected.

4. If the record(s) already exist in that file, a message box will appear asking if you want them to be overwritten (Figure 4.22). If the number of pathnames is four or less, their names will be displayed in the message box; otherwise they will be printed in the log screen.

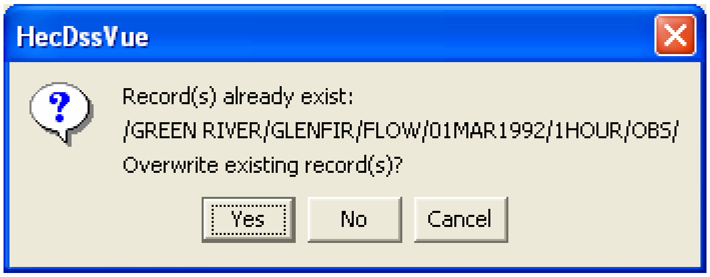


Look in: BaldEagleDemo

File name: *.dss

Files of type: All Files (*.*)

Figure 4.21 HEC-DSSVue Copy Records into HEC-DSS File Dialog Box



Record(s) already exist:
/GREEN RIVER/GLENFIR/FLOW/01MAR1992/1HOUR/OBS/
Overwrite existing record(s)?

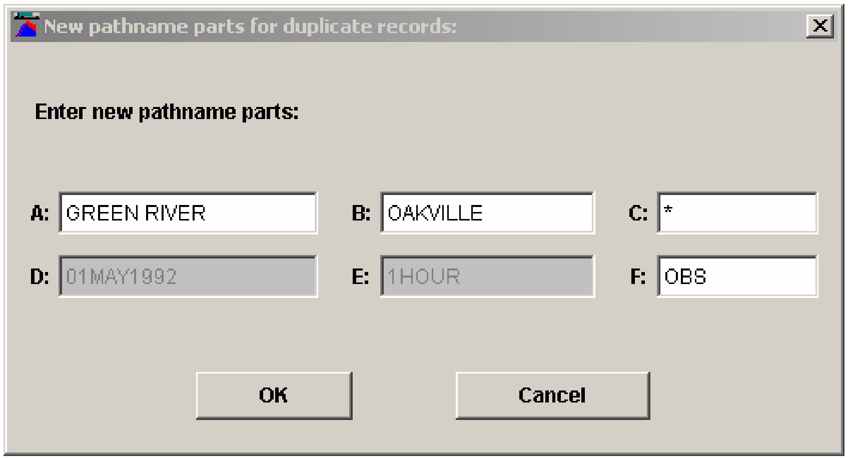
Figure 4.22 Message Asking if Records Should be Overwritten.

4.8 Duplicating Records

The **Duplicate** command duplicates records in the same HEC-DSS file, giving the new records different pathnames.

To duplicate records:

1. Select the record or records to duplicate.
2. From the Utilities menu, select **Duplicate Records**. The **New Pathname Parts for Duplicate Records** dialog box (Figure 4.23) will open.



Enter new pathname parts:

A: GREEN RIVER B: OAKVILLE C: *

D: 01MAY1992 E: 1HOUR F: OBS

Figure 4.23 HEC-DSSVue New Pathname Parts for Duplicate Records Dialog Box

3. Type the new **Pathname Parts** into the A, B, C, D, E, and F boxes You cannot change the D or E parts for time series data (use Math Functions to accomplish this). For multiple records, pathname parts that are the same for all records will show up in the dialog. Where parts differ (such as having pathnames with C parts of FLOW and ELEVATION), the part will be displayed with an asterisk (*).
4. Click **OK**. A confirmation message will appear, stating that the records have been duplicated.

4.9 Deleting Records

When you delete data from a HEC-DSS file, the data is marked as missing, but not actually removed until you squeeze the file. You can recover deleted records with the **Undelete** command, described later. Once a file has been squeezed, records cannot be undeleted. To delete records from a HEC-DSS file:

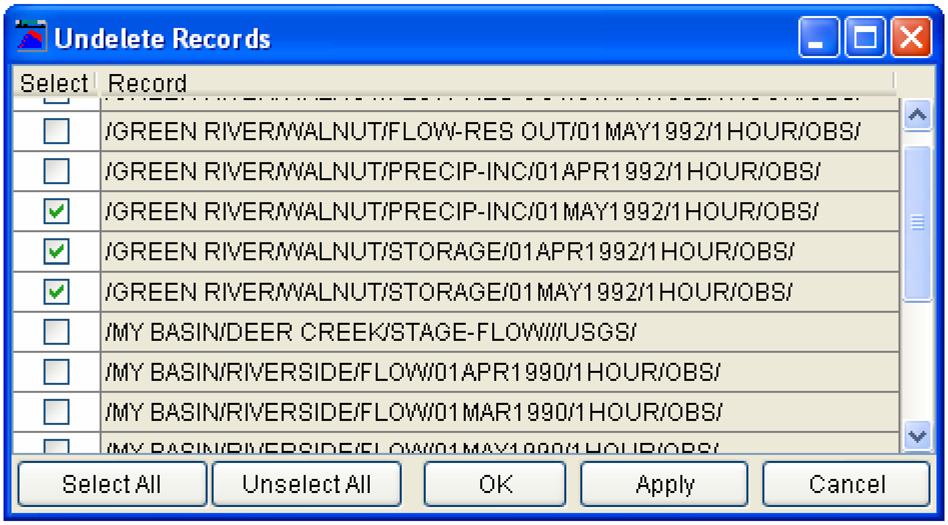
1. Select the record or records to delete.
2. From the Utilities menu, select **Delete Records**. A dialog box will confirm that you want to delete those records. If the number or records is four or less, their pathnames will be displayed in the dialog.
3. Click **OK**. A confirmation message will appear, stating that the records have been deleted.

4.10 Undoing Deletions

Deleted records may be recovered as long as the HEC-DSS file has not been squeezed. When a file is squeezed, all deleted records are physically removed.

You can undelete records in three ways: 1) Undelete all records in the file; 2) Select the records to undelete from a list; 3) Undelete the records that you just deleted. To undelete records from a HEC-DSS file:

1. From the Utilities menu, select **UnDelete** and then select either **All**, **Selected**…, or **Last Deleted**.
2. To undelete all records select **All**. A dialog box will confirm that you want to undelete all of the records in the file. Click OK. A confirmation message will appear, stating that the records have been undeleted.
3. To undelete the records that you have just deleted, select **Last Deleted**. A dialog box will confirm that you want to undelete the records that you just deleted. Click OK. A confirmation message will appear, stating that the records have been undeleted.
4. If you choose **Selected…**, a list of the pathnames that you can undelete will appear (Figure 4.24). Check the boxes of the records that you want to undelete, or press the **Select All** button to undelete all records in the list. You can unselect the checked records by pressing the **Unselect All** button. Once you have made your selection, press the **OK** button to undelete those records and close the dialog, or press **Apply** to undelete the records and leave the dialog open, or press **Cancel** to quit the dialog. A confirmation message will appear stating that the records have been undeleted.



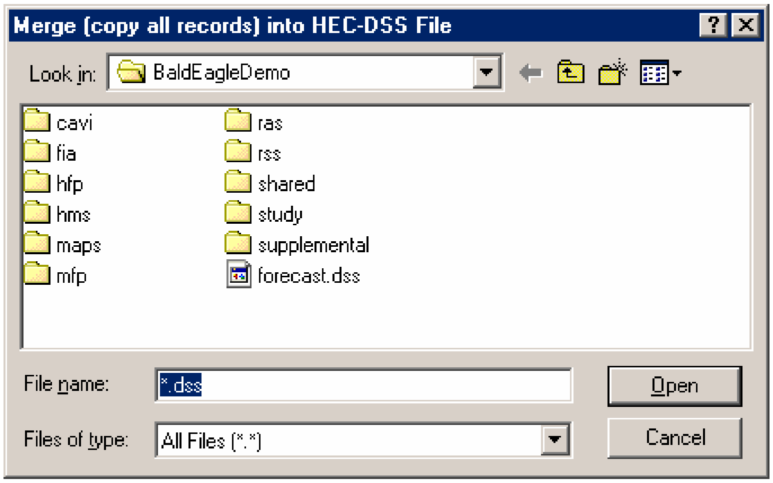
Select	Record
□	/GREEN RIVER/WALNUT/FLOW-RES OUT/01MAY1992/1HOUR/OBS/
□	/GREEN RIVER/WALNUT/PRECIP-INC/01APR1992/1HOUR/OBS/
☑	/GREEN RIVER/WALNUT/PRECIP-INC/01MAY1992/1HOUR/OBS/
☑	/GREEN RIVER/WALNUT/STORAGE/01APR1992/1HOUR/OBS/
☑	/GREEN RIVER/WALNUT/STORAGE/01MAY1992/1HOUR/OBS/
□	/MY BASIN/DEER CREEK/STAGE-FLOW///USGS/
□	/MY BASIN/RIVERSIDE/FLOW/01APR1990/1HOUR/OBS/
□	/MY BASIN/RIVERSIDE/FLOW/01MAR1990/1HOUR/OBS/
□	/MY BASIN/RIVERSIDE/FLOW/01MAY1990/1HOUR/OBS/

Figure 4.24 HEC-DSSVue Undelete Records Selection Dialog Box

4.11 Merging HEC-DSS Files

Merging copies all of the records in the currently opened HEC-DSS file into another. This is similar to selecting all records then using the **Copy Records** option, but is much more efficient. However, this option will overwrite any records with the same pathnames and will not splice together time series records. To merge the current HEC-DSS file into another:

1. From the **Utilities** menu, click **Merge Files**. The **Merge (copy all records) into HEC-DSS File** dialog box (Figure 4.25) will open.
2. In the **File Name** box, select an existing HEC-DSS File into which you want to copy all of the records into and click **Open**. A confirmation message will appear stating that the records have been copied into the HEC-DSS file you selected.


Merge (copy all records) into HEC-DSS File

Look in: BaldEagleDemo

cavi, fia, hfp, hms, maps, mfp, ras, rss, shared, study, supplemental, forecast.dss

File name: *.dss

Files of type: All Files (*.*)

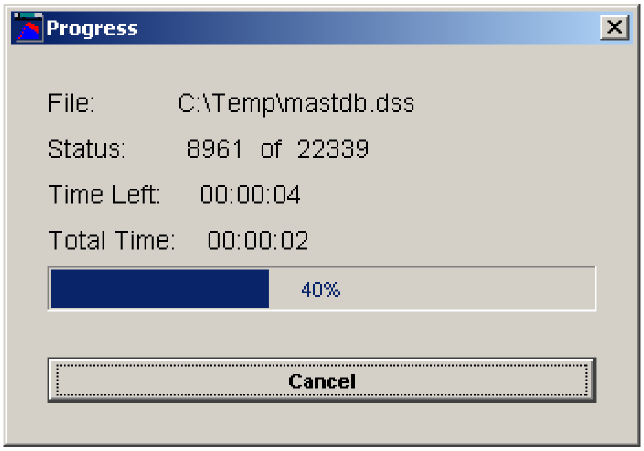
Figure 4.25 HEC-DSSVue Merge (copy all records) into HEC-DSS File Dialog Box

4.12 Squeezing HEC-DSS Files

When you delete or rename records, a HEC-DSS file will accumulate inactive space. The **Squeeze** command removes inactive space by copying all valid data to a new file then renaming the new file to the old filename. The **Squeeze** command will also automatically re-adjust internal HEC-DSS table sizes to optimize access to the data. This is similar to de-fragmenting your file system. Once a squeeze has been accomplished, deleted data cannot be recovered.

To squeeze an HEC-DSS file:

1. Open the HEC-DSS file, then, from the **Utilities** menu, click **Squeeze**. A window will appear indicating the status of the squeeze process, as shown in Figure 4.26.
2. If you wish, you may cancel the squeeze prior to completion, by pressing the **Cancel** button.
3. When the process is complete, a confirmation will appear (Figure 4.27)



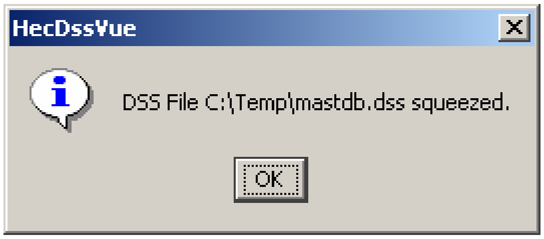
File: C:\Temp\mastdb.dss

Status: 8961 of 22339

Time Left: 00:00:04

Total Time: 00:00:02

Figure 4.26 HEC-DSSVue Squeeze Progress Window

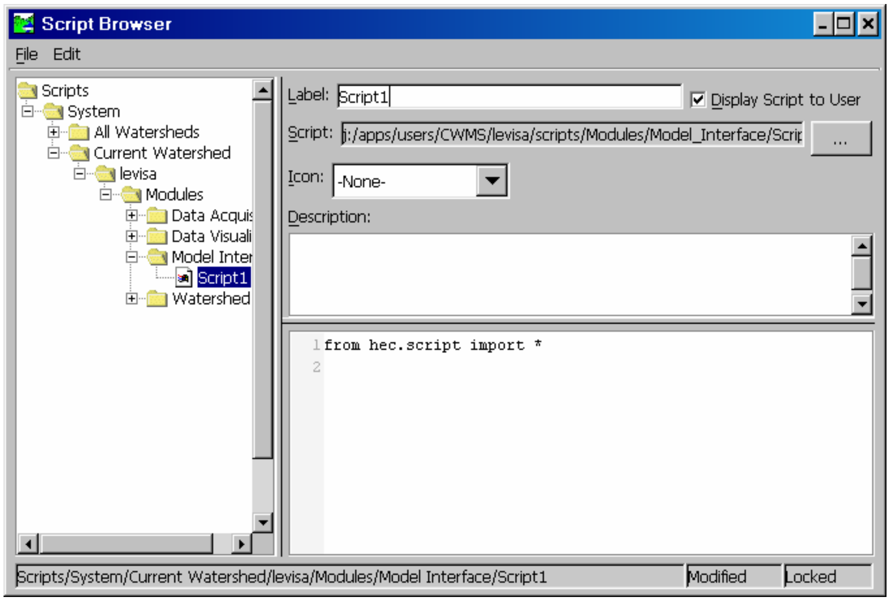


DSS File C:\Temp\mastdb.dss squeezed.

Figure 4.27 Example HEC-DSSVue Squeeze Confirmation Message

4.13 Scripting

You can use Jython scripts that you or others write, to perform math functions, create custom plots or tables, and automate repetitive tasks. To edit, test, and select scripts, use the **Script Browser**. To run scripts that have already been setup, use the **Script Selector** window. To access the **Script Browser** (Figure 4.28), from the **Utilities** menu, click **Script Browser**. To access the **Script Selector** window, from the **Utilities** menu, click **Script Selector**. For more information, refer to the chapter on **Scripting**.



Label: Script1

☑ Display Script to User

Script: j:/apps/users/CWMS/levisa/scripts/Modules/Model_Interface/Scrip

Icon: -None-

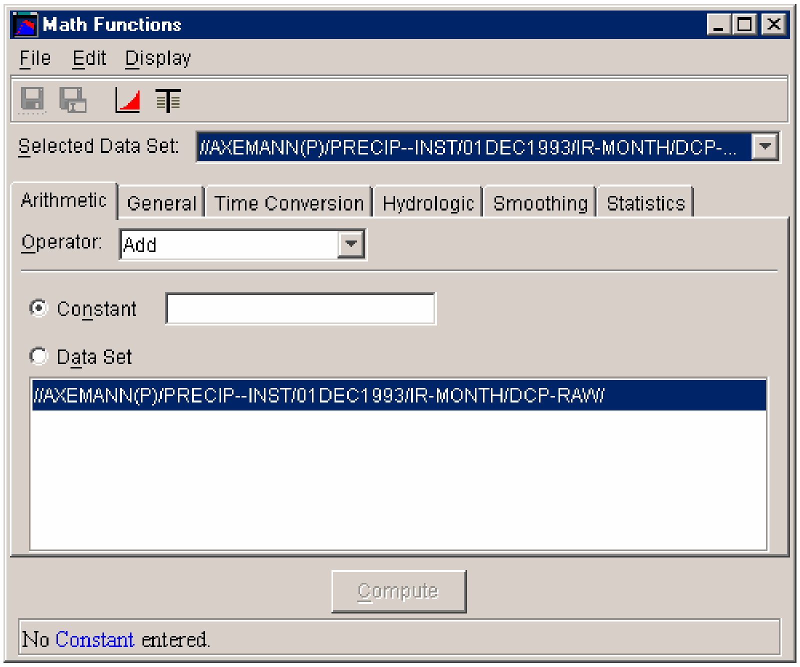
```
from hec.script import *

```

Scripts/System/Current Watershed/levisa/Modules/Model Interface/Script1 | Modified | Locked

Figure 4.28 Script Browser

4.14 Performing Math Functions

To access HEC-DSSVue Math Functions, from the **Utilities** menu, click **Math Functions**. The **Math Functions** dialog box (Figure 4.29) will open.



Selected Data Set: //AXEMANN(P)/PRECIP--INST/01DEC1993/IR-MONTH/DCP-...

Operator: Add

● Constant

○ Data Set

//AXEMANN(P)/PRECIP--INST/01DEC1993/IR-MONTH/DCP-RAW/

No Constant entered.

Figure 4.29 HEC-DSSVue Math Functions Dialog Box

HEC-DSSVue Math Functions facilitate the mathematical manipulation of time series and paired data sets you have selected. Available functions fall into six categories: **Arithmetic**, **General**, **Time Conversion**, **Hydrologic**, **Smoothing**, and **Statistics**. Each function category is a tab on the Math Functions screen.

Use the **Operator** list to select functions, and choose data sets from the **Selected Data Set** list. Click the **Compute** button to apply a function. The Compute button is available only if the data on the function screen is complete.

Refer to the Chapter 6 on **Math Functions** for more information.

Chapter 5

Customizing Plots

Plots are highly customizable and offer an array of information that will assist you with reviewing your data. Figure 5.1 shows a sample plot illustrating raw and revised data for stage and flow at a location called Beech Creek Station.

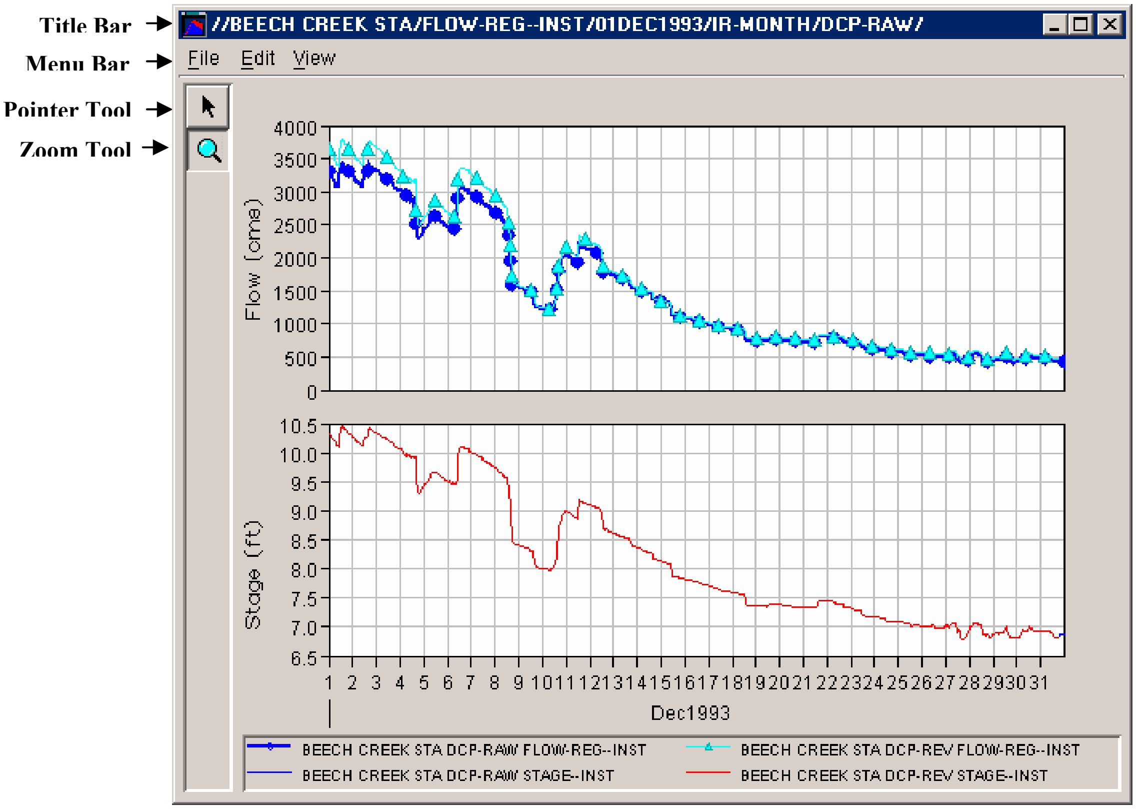


Figure 5.1 Plot Window

With the **Pointer Tool** , you can access shortcut menus that allow you to customize features of your plots using the plot window's editing tools. The following sections discuss these tools in detail.

The **Zoom Tool** allows you to view data closely at a specific time. To zoom in, select the Zoom Tool then "draw" a rectangle around the section of the plot you wish to enlarge. To zoom out, right-click anywhere in the display area using the Zoom Tool.

If you wish to keep the plot window on top of your desktop so you can view it while working in other windows, you can select **Always on Top** from the **View** menu. A check mark indicates this option is active.

5.1 Customizing Plots: Overview

Plot properties editors allow you to configure default properties for plots as well as customize individual plots.

Figure 5.2 shows the features of plots you can configure.

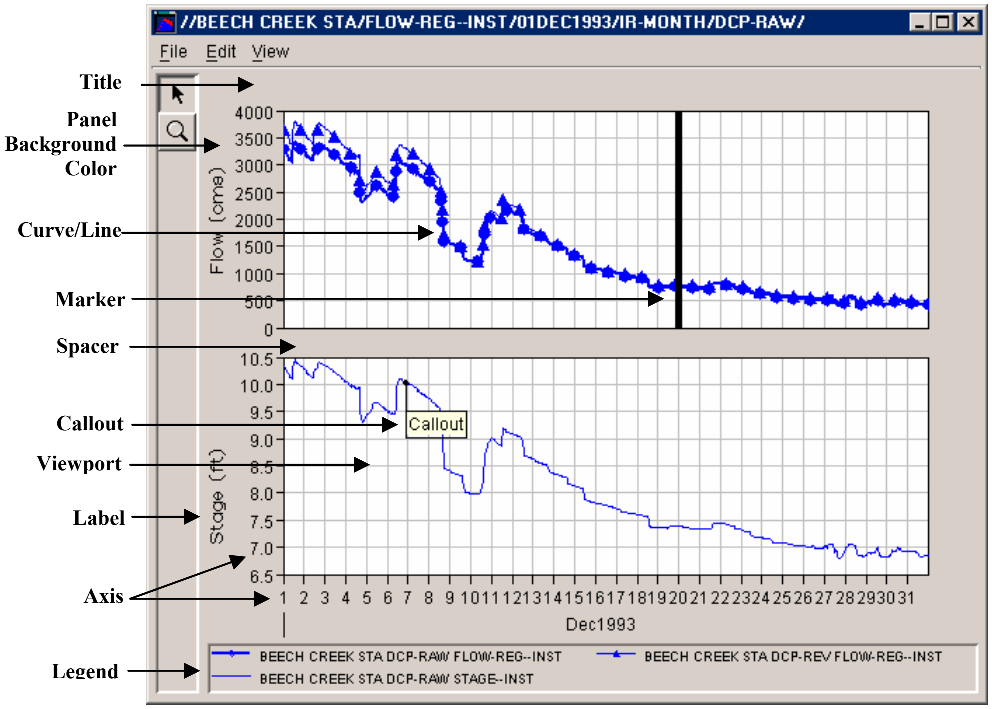


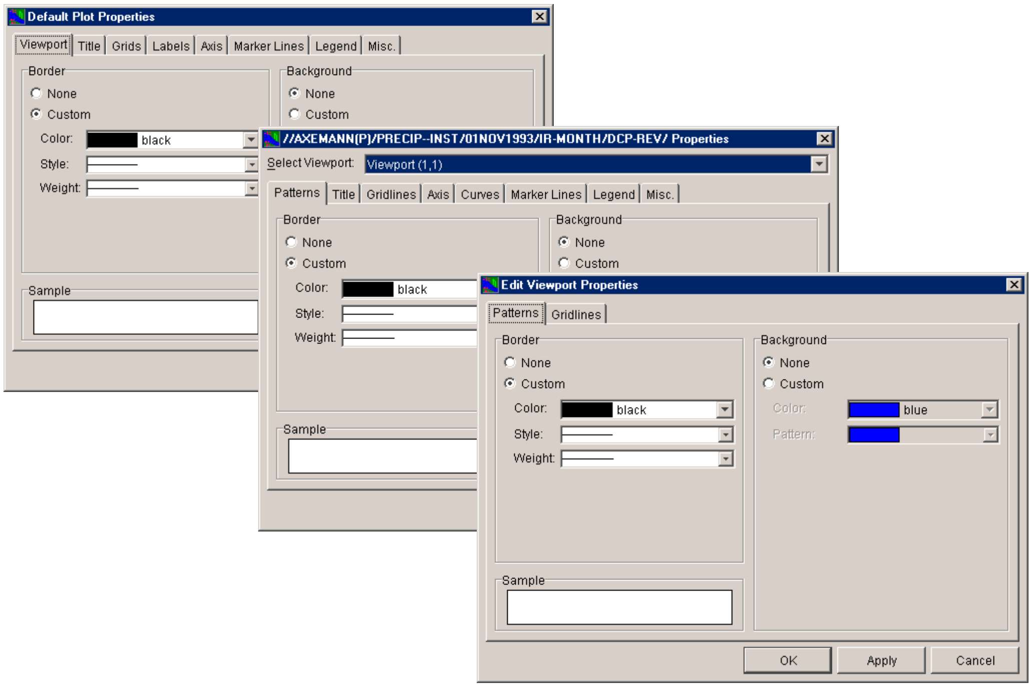
Figure 5.2 Configurable Features of Plots

- **Title**: Optionally, you can add a title to the plot display.
- **Panel Background Color**: You can specify the background color of the plot window (light grey is the default).
- **Curve/Line Properties**: You can choose the line and point styles, add labels, and specify symbols to indicate quality.
- **Marker:** You can add markers on the X and Y axes and customize the appearance of these markers.
- **Spacer:** You can specify the distance between viewports, between a viewport and the legend, and the width of side margins.
- **Callout:** You can add descriptive callouts at specific points along a line.
- **Viewport:** You can customize the border around the viewport, the background color and pattern, and the appearance of gridlines. You can also specify the number, size, and content of viewports.
- **Label:** You can add borders and backgrounds to axis and legend labels.
- **Axis:** You can specify either a linear or log axis type, specify the axis scale, and customize tic marks.

- **Legend:** You can add titles to the plot legend and specify whether the legend appears below or to the right of the plot.

5.2 Using Plot Editors

Several different editing interfaces allow you to either set defaults for all plots or specify properties of individual plots. Figure 5.3 shows three examples: the **Default Plot Properties Editor**, the **Plot Properties Editor**, and the **Viewport Properties Editor**.


Default Plot Properties

Border: ○ None ● Custom
Color: black

Background: ● None ○ Custom

//AXEMANN(P)/PRECIP--INST/01NOV1993/IR-MONTH/DCP-REV/ Properties

Select Viewport: Viewport (1,1)

Border: ○ None ● Custom
Color: black

Background: ● None ○ Custom

Edit Viewport Properties

Border: ○ None ● Custom
Color: black

Background: ● None ○ Custom
Color: blue

Viewport | Title | Grids | Labels | Axis | Marker Lines | Legend | Misc.
Style:
Weight:
Sample
Patterns | Title | Gridlines | Axis | Curves | Marker Lines | Legend | Misc.
Patterns | Gridlines
Pattern:
OK
Apply
Cancel

Figure 5.3 Examples of Plot Properties Editors

As Figure 5.3 illustrates, the editors can look almost identical to each other. Therefore, it is necessary for you to understand what the editors do and how to access the correct editor for your purpose.

5.2.1 Setting Defaults vs. Customizing Individual Plots

Plot editors and tools fall into two categories in terms of function: either they allow you to specify *defaults* for *all* plots you create, or they allow you to customize *individual* plots.

Across these two functional categories, the plot editors and tools either allow you to edit a variety of plot properties, or they can be specialized editors that allow you to edit a single property.

To configure the *default* appearance of *all* plots, you need to use the **Default Plot Properties Editor** and **Default Line Style Options Editor**. All settings you specify in these editors will apply to all new plots you open. You can access these editors only from the **Edit** menu of plots.

To customize *individual* plots, you can use the **Plot Properties Editor** and the **Configure Plot Editor**, both accessed from the **Edit** menu of a plot window. Also, using right-click shortcut menus, you can access specialized editors for individual plot features. The **Viewport Properties Editor** is an example of a specialized editor.

Additionally, once you have customized an individual plot, you can export its settings as a **Template** that you can apply to other plots. To create a template based upon a plot, you will use the **Export Properties** option in the **File** menu of the plot window.

Likewise, you can import **Templates** to apply previously defined properties to an individual plot. To apply a template to a new plot, you will use the **Import Properties** option in the **File** menu of the new plot window.

5.2.2 Accessing Editors

You can access properties editors from **shortcut menus** (Figure 5.4) and from the **Edit** menu in the plot window (Figure 5.5).

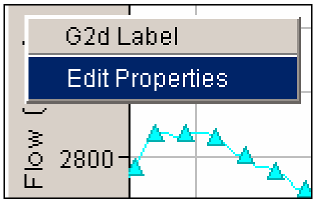


Figure 5.4 Shortcut Menu

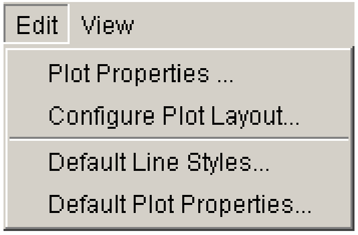


Figure 5.5 Edit Menu

You will use these menus according to whether you are defining properties for an individual plot or setting default properties for all plots you create.

Use **shortcut menus** to edit specific components of an individual plot. For example, if you want to edit axis label properties on a plot, right-click on the axis label to access the shortcut menu for the label (Figure 5.4), then select the **Edit Properties** command for the label.

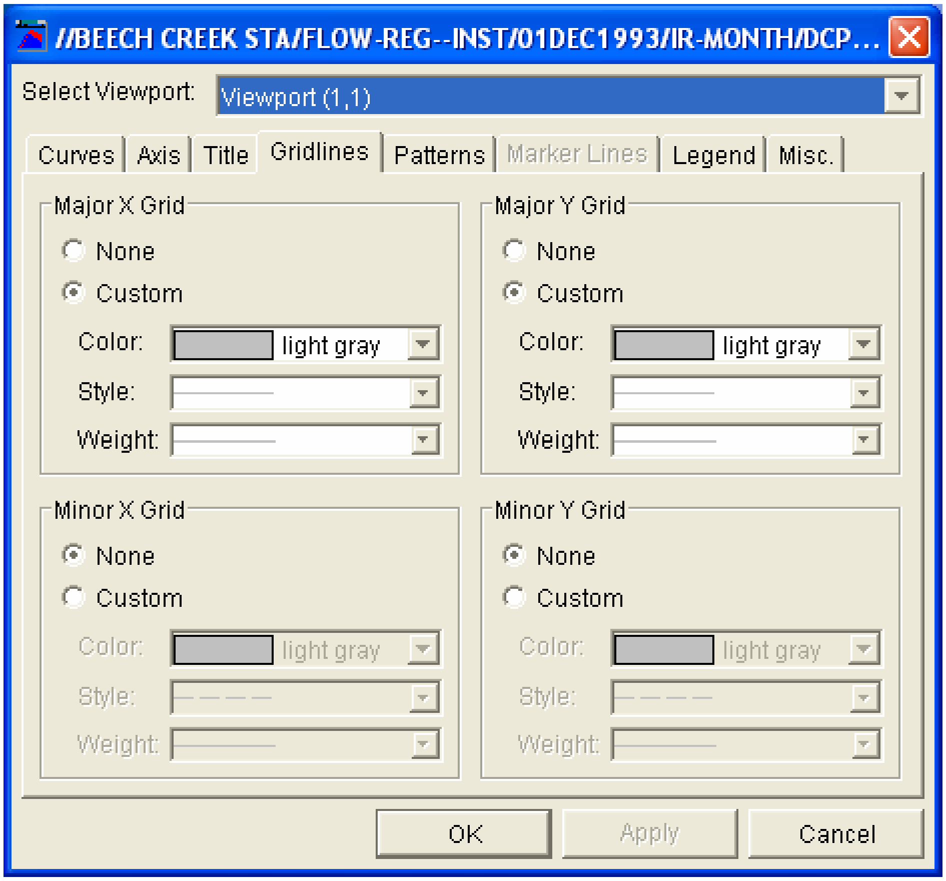
Use the **Edit** menu of a plot window to access the **Plot Properties Editor**, **Default Line Style Options Editor**, **Default Plot Properties Editor**, and **Configure Plot Editor**. These editors, discussed in Section 5.3, allow you to edit a variety of plot properties.

5.3 Recognizing Plot Editors and Tools

Following is an overview of the editing tools that allow you to customize plots. Later sections provide more detailed instructions on editing specific plot properties using these tools.

5.3.1 Plot Properties Editor

The **Plot Properties Editor** (Figure 5.6) is accessed from the **Edit** menu of a plot, and allows you to configure multiple display properties of an individual plot, including the **Curves**, **Axis**, the plot **Title**, **Gridlines**, border and background **Patterns** of the viewport, **Marker Lines**, **Legend,** and properties of the plot window panel.


//BEECH CREEK STA/FLOW-REG--INST/01DEC1993/IR-MONTH/DCP...

Select Viewport: Viewport (1,1)

Major X Grid: ○ None ● Custom; Color: light gray; Style; Weight

Major Y Grid: ○ None ● Custom; Color: light gray; Style; Weight

Minor X Grid: ● None ○ Custom; Color: light gray; Style; Weight

Minor Y Grid: ● None ○ Custom; Color: light gray; Style; Weight

Figure 5.6 Plot Properties Editor

When you customize properties of a plot using the Plot Properties Editor, your changes apply only to that individual plot unless you export the plot's properties (see Section 5.12).

To access the Plot Properties Editor, from the **Edit** menu, choose **Plot Properties**.

5.3.2 Individual Plot Property Editors

When you want to edit a specific property of a plot without launching the Plot Properties Editor (discussed in Section 5.3.1), you can use individual plot property editors instead. These individual plot property editors correspond to the tabs of the Plot Properties Editor.

To access an individual plot property editor, right-click on the element you want to edit, then select **Edit Properties** from the shortcut menu.

For example, if you right-click inside the gridded plot area, called the *viewport*, you will see the **Viewport shortcut menu** (Figure 5.7). When you choose **Edit Properties**, the **Viewport Properties Editor** will open (Figure 5.8).

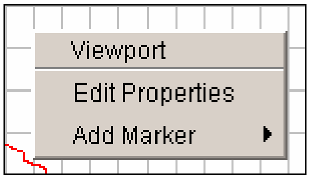
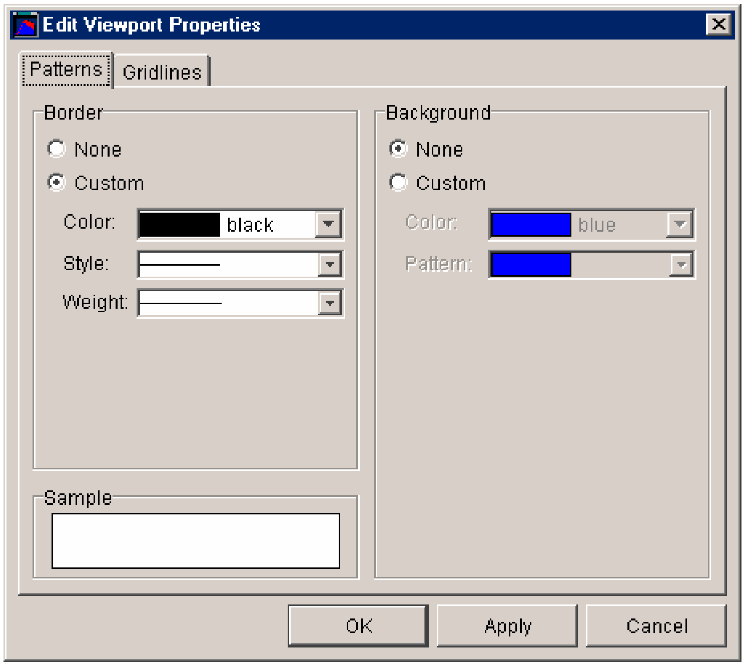


Figure 5.7 Viewport Shortcut Menu

The **Viewport Properties Editor** lets you edit only properties associated with the viewport, using the same **Patterns** and **Gridlines** tabs as appear in the Plot Properties Editor.


Edit Viewport Properties

Patterns | Gridlines

Border
○ None
● Custom
Color: black
Style:
Weight:

Background
● None
○ Custom
Color: blue
Pattern:

Sample

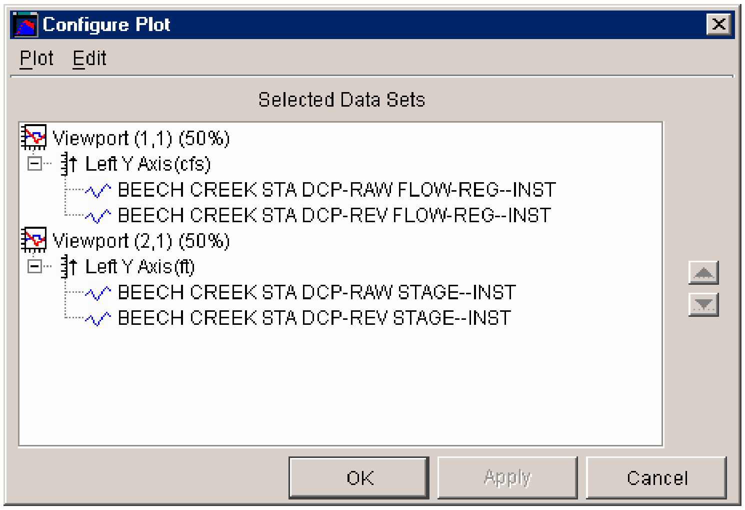
Figure 5.8 Viewport Properties Editor

Other individual properties editors are the **Title Properties Editor**, **Axis Properties Editor, Curve Properties Editor**, **Label Properties Editor**, **Marker Properties Editor**, **Legend Properties Editor**, and the **Spacer Properties Editor**.

The only plot property you cannot edit using an individual property editor is the color of the plot window panel. (See Section 5.10 for more information.)

5.3.3 Configure Plot Editor

The **Configure Plot Editor** (Figure 5.9) is accessed from the **Edit** menu of a plot and allows you to customize the layout of an individual plot. You can add and remove axes and add, remove, arrange the order of, and set the weight of viewports in the plot window.



Viewport (1,1) (50%)
Left Y Axis(cfs)
BEECH CREEK STA DCP-RAW FLOW-REG--INST
BEECH CREEK STA DCP-REV FLOW-REG--INST
Viewport (2,1) (50%)
Left Y Axis(ft)
BEECH CREEK STA DCP-RAW STAGE--INST
BEECH CREEK STA DCP-REV STAGE--INST

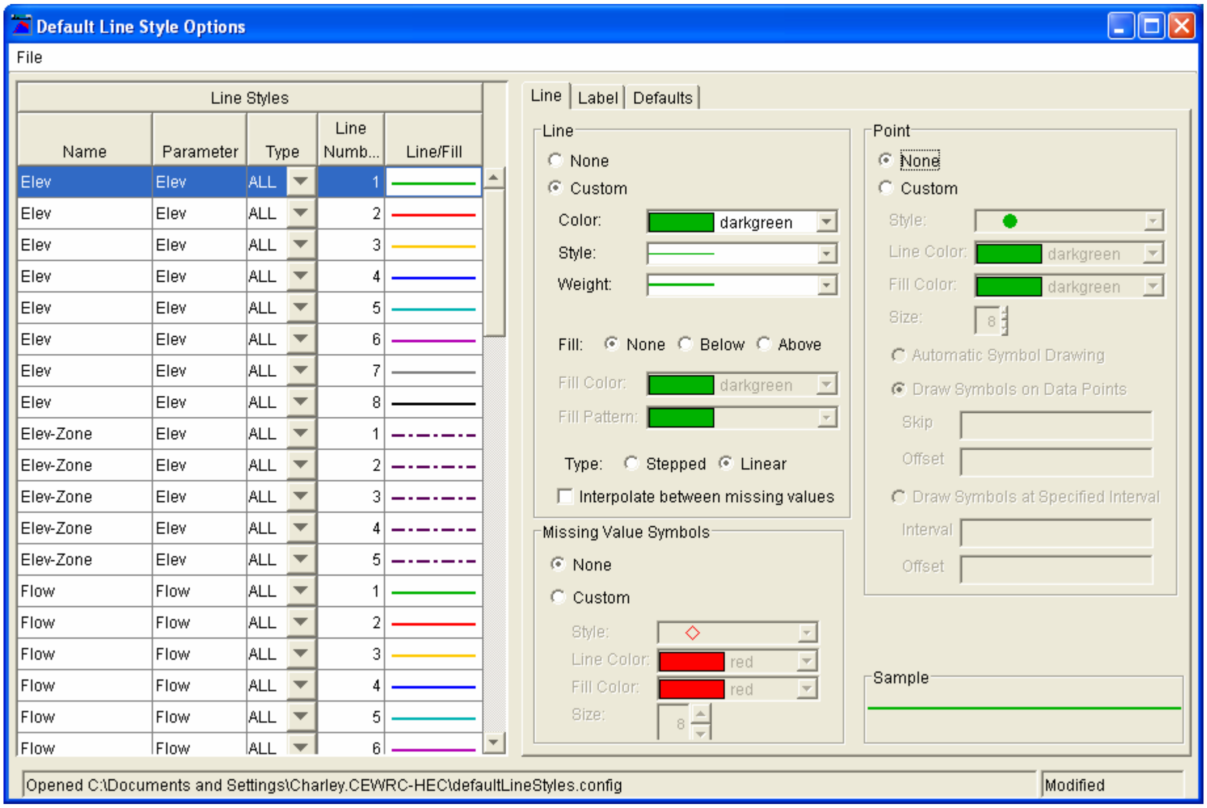
Figure 5.9 Configure Plot Editor

When you customize the layout of a plot using the Configure Plot Editor, your changes apply only to that individual plot unless you export the plot's properties (see Section 5.12).

To access the Configure Plot Editor, from the **Edit** menu, click **Configure Plot Layout**.

5.3.4 Default Line Style Options Editor

With the **Default Line Style Options Editor** (Figure 5.10), you can specify the default line and fill styles, as well as labels, used across all plots for specific parameters.



Line Styles

Name	Parameter	Type	Line Numb...	Line/Fill
Elev	Elev	ALL	1	———
Elev	Elev	ALL	2	———
Elev	Elev	ALL	3	———
Elev	Elev	ALL	4	———
Elev	Elev	ALL	5	———
Elev	Elev	ALL	6	———
Elev	Elev	ALL	7	———
Elev	Elev	ALL	8	———
Elev-Zone	Elev	ALL	1	-·-·-·-
Elev-Zone	Elev	ALL	2	-·-·-·-
Elev-Zone	Elev	ALL	3	-·-·-·-
Elev-Zone	Elev	ALL	4	-·-·-·-
Elev-Zone	Elev	ALL	5	-·-·-·-
Flow	Flow	ALL	1	———
Flow	Flow	ALL	2	———
Flow	Flow	ALL	3	———
Flow	Flow	ALL	4	———
Flow	Flow	ALL	5	———
Flow	Flow	ALL	6	———

Line
○ None
● Custom
Color: 🟩 darkgreen
Style: ———
Weight: ———
Fill: ● None ○ Below ○ Above
Fill Color: 🟩 darkgreen
Fill Pattern: 🟩
Type: ○ Stepped ● Linear
□ Interpolate between missing values

Missing Value Symbols
● None
○ Custom
Style: ◇
Line Color: 🟥 red
Fill Color: 🟥 red
Size: 8

Point
● None
○ Custom
Style: ●
Line Color: 🟩 darkgreen
Fill Color: 🟩 darkgreen
Size: 8
○ Automatic Symbol Drawing
● Draw Symbols on Data Points
Skip
Offset
○ Draw Symbols at Specified Interval
Interval
Offset

Sample

Opened C:\Documents and Settings\Charley.CEWRC-HEC\defaultLineStyles.config | Modified

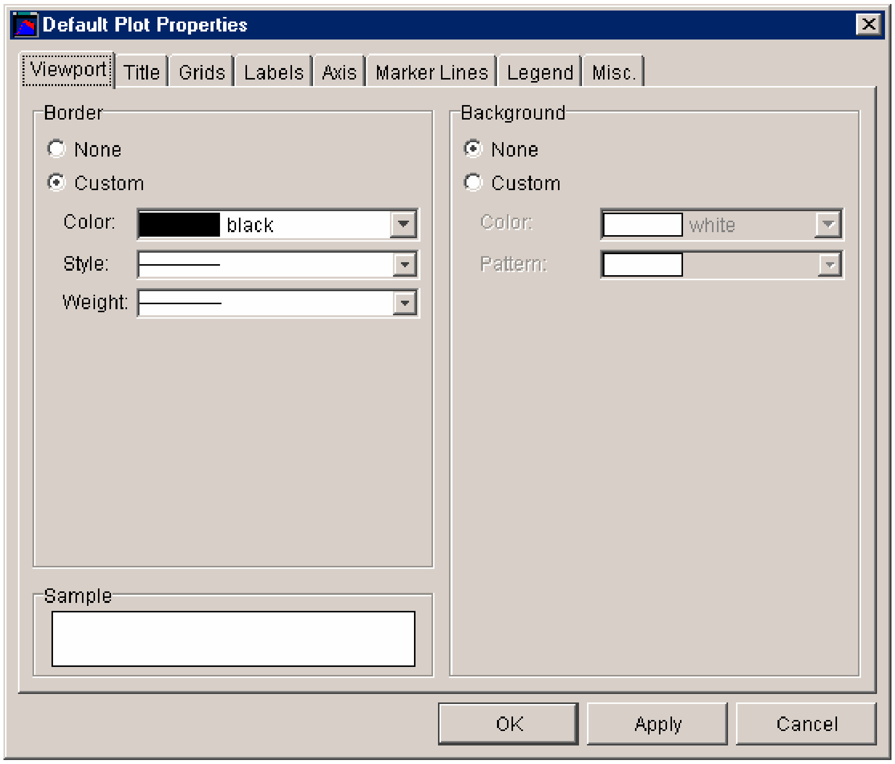
Figure 5.10 Default Line Style Options Editor

To access the Default Line Style Options Editor, from the **Edit** menu, choose **Default Line Styles**.

Refer to Section 5.5.2 for more information on the Default Line Styles Options Editor.

5.3.5 Default Plot Properties Editor

The **Default Plot Properties Editor** (Figure 5.11) allows you to configure the default display properties of all plots you create. Properties you can configure include the border and background of the **Viewport,** the **Title**, **Grid lines**, **Labels**, **Axis**, **Marker Lines**, **Legend,** and miscellaneous properties of the plot window panel.



Border

○ None

● Custom

Color: black

Style:

Weight:

Sample

Background

● None

○ Custom

Color: white

Pattern:

Figure 5.11 Default Plot Properties Editor

When you customize plot properties using the Default Plot Properties Editor, your changes apply to all plots you create.

To access the Default Plot Properties Editor, from the **Edit** menu, click **Default Plot Properties**.

5.4 Customizing Plot Titles

You can add titles to individual plots and configure default properties for all plot titles.

To add or edit a title on an *individual* plot, you can either:

- From the **Edit** menu, choose **Plot Properties**. When the **Plot Properties Editor** opens, select the **Title** tab.

  Or

- Right-click in the blank area above the plot (below the menu bar) with the **Pointer Tool**, and then select **Edit Properties** from the shortcut menu (Figure 5.12). The **Title Properties Editor** will open.

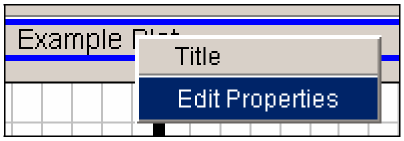
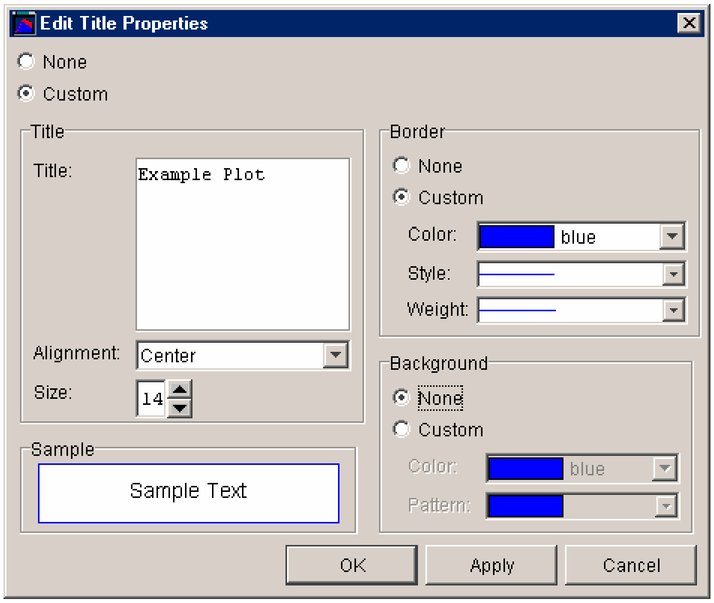


Figure 5.12 Shortcut Menu--Title Properties

To specify the appearance of titles for *all* of your plots, from the **Edit** menu, click **Default Plot Properties**, then select the **Title** tab of the **Default Plot Properties** editor.

Whether you are using the **Plot Properties Editor**, the specialized **Title Properties Editor**, or the **Default Plot Properties Editor**, the worksheet for editing plot title properties is the same.

Figure 5.13 shows the **Title Properties Editor**. This editor contains the same fields as the **Title** tab of the **Plot Properties Editor** and the **Default Plot Properties Editor**.



○ None
● Custom

Title
Title: Example Plot
Alignment: Center
Size: 14

Border
○ None
● Custom
Color: blue
Style:
Weight:

Background
● None
○ Custom
Color: blue
Pattern:

Sample
Sample Text

Figure 5.13 Edit Title Properties

1. To specify a title for the plot, select **Custom**.
2. In the **Title** panel, type the title you want in the **Title** box.
3. From the **Alignment** list, select the alignment for the plot title. Your choices are Center, Left, or Right.
4. From the **Size** list, select the text size for the title. The **Sample** box provides a preview of your plot title.
5. The **Border** group allows you to add a border around the title. You can specify the **Color**, line **Style**, and line **Weight**.
6. The **Background** group lets you add a background **Color** and/or **Pattern** behind your plot title.

Click **Apply** to save your changes and continue adjusting the appearance of the title. Click **OK** when you are finished.

5.5 Customizing Curves

You can customize line and point styles, add labels, and specify symbols to indicate data quality in your plots. Additionally, you can specify the parameter-based default curve styles used across all plots.

There are three different ways to edit plot curves, depending on whether you wish to customize one or more curves in an individual plot or specify defaults for all plots.

5.5.1 Customizing Curves in Individual Plots

To customize *all* curves in an *individual* plot, from the **Edit** menu, choose **Plot Properties**. When the **Plot Properties Editor** opens, select the **Curves** tab.

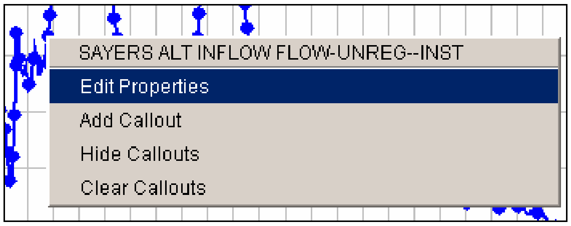


Figure 5.14 Shortcut Menu--Curve Properties

To customize a *specific* curve in an *individual* plot, right-click on the line or curve you wish to edit with the **Pointer Tool**, then select **Edit Properties** from the shortcut menu (Figure 5.14). The **Edit Curve Properties Editor** will open.

The **Curves** tab of the **Plot Properties Editor** and the **Edit Curve Properties Editor** are nearly identical with two exceptions:

- First, at the top of the **Curves** tab of the **Plot Properties Editor**, there is a list of all curves contained in the plot. In contrast, when you open the **Edit Curve Properties Editor**, it deals only with the curve you selected when you launched the editor; therefore, there is no list of curves.

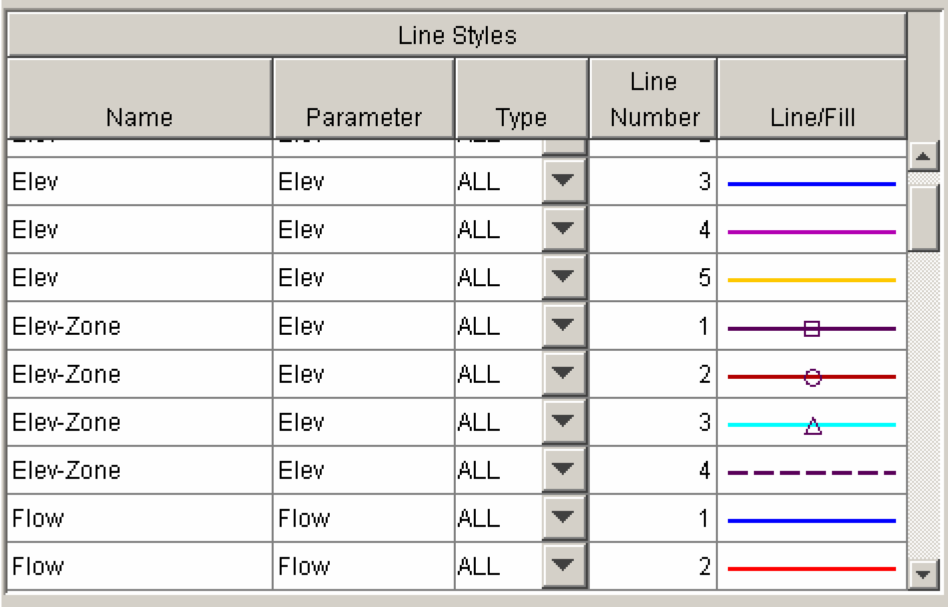
- Second, the **Curves** tab of the **Plot Properties Editor** has a **Remove Line** button, whereas the **Edit Curve Properties Editor** does not.

Despite these differences, both the **Edit Curve Properties Editor** and the **Curves** tab of the **Plot Properties Editor** allow you to edit **Style**, **Label**, and **Quality Symbols**. (You can edit Quality Symbols only if the plot has quality set for its data.)

5.5.2 Specifying Parameter-Based Default Curve Styles

To specify parameter-based default curve styles for *all* of your plots, from the **Edit** menu, click **Default Line Styles**. The **Default Line Styles Options Editor** will open (Figure 5.10).

The **Default Line Styles Options Editor** gives you a way to edit line styles from the **Line Styles** box (Figure 5.15). You can specify the default, parameter-based curve styles used for all plots.



Line Styles

Name	Parameter	Type	Line Number	Line/Fill
Elev	Elev	ALL	3	────
Elev	Elev	ALL	4	────
Elev	Elev	ALL	5	────
Elev-Zone	Elev	ALL	1	──□──
Elev-Zone	Elev	ALL	2	──○──
Elev-Zone	Elev	ALL	3	──△──
Elev-Zone	Elev	ALL	4	- - - -
Flow	Flow	ALL	1	────
Flow	Flow	ALL	2	────

Figure 5.15 Default Line Style Options Editor: Detail of Line Styles Box

The Line Styles box displays typical data types with default line and fill styles predefined.

You can edit all of these fields, change default line and fill styles for existing types, and add new data types to the list. At this time, there is no delete option.

Name and Parameter
To edit an existing name or parameter, highlight it, and then enter the new name or parameter. The name corresponds to the "C" part of HEC-DSS pathnames. The parameter associates data sets to be plotted in the same viewport. For example, FLOW-IN and FLOW-OUT are different "C" parts, but both are FLOW data sets and are to be plotted in the same viewport.

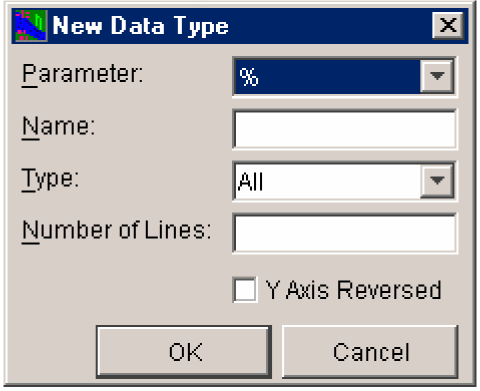
Type
To change the data type associated with a Name and Parameter, click the down-arrow and select from the list.

Line Number
The Line Number column indicates the number of lines associated with a data type. See **Adding New Data Styles** below.

Line/Fill
The Line/Fill property determines how curves associated with a particular name/parameter/type combination will appear in all plots. To specify the Line/Fill, select the row, and then customize the Line and Point properties as discussed in Section 5.5.3.

Adding New Data Styles
To add a new data type:

1. From the **File** menu of the **Default Line Style Options Editor,** choose **New**. The **New Data Type** dialog box will open (Figure 5.16).
2. Select a parameter from the **Parameter** list.
3. Enter a name in the **Name** box.
4. From the **Type** list, select the data type.
5. In the **Number of Lines** box, enter the number of curves you want to add for this new data type.
6. If you want to reverse the Y-axis, select **Y Axis Reversed**.
7. Click **OK** to close the dialog box.



Parameter: %

Name:

Type: All

Number of Lines:

□ Y Axis Reversed

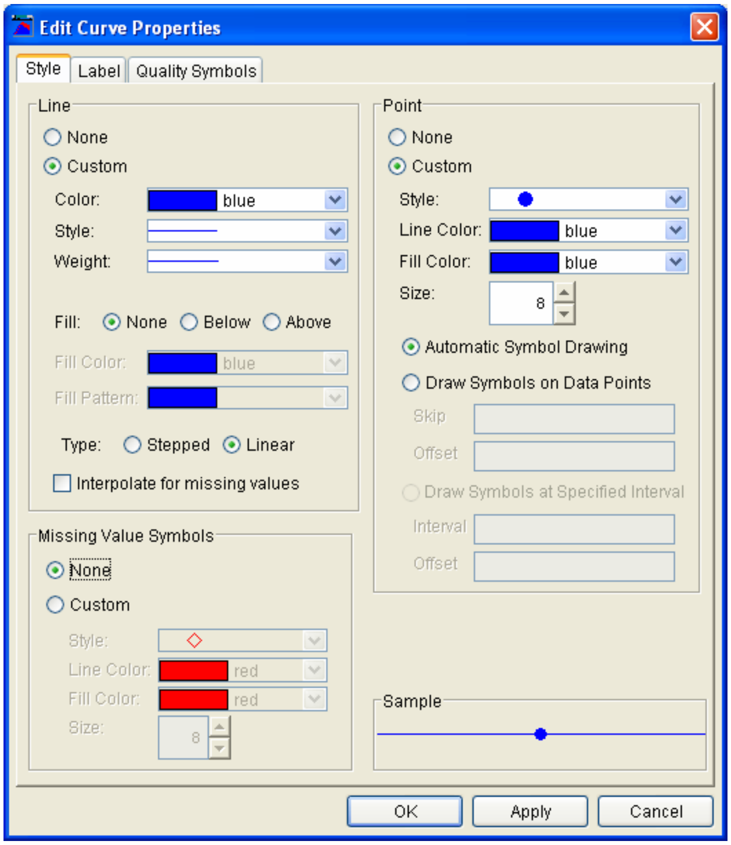
Figure 5.16 New Data Type Dialog Box

The Line Styles box will now display the new data type you have added, repeated as many times as you specified in the Number of Lines box (reflected in the **Line Number** column). You can customize the new data types as described above.

To save your changes, from the **File** menu, click **Save**. To close the Default Line Styles Options Editor, from the **File** menu, click **Close**.

5.5.3 Specifying Line and Point Styles of Curves

Figure 5.17 shows the curve Line and Point **Style** worksheet. This worksheet is available from the **Curves** tabs of the **Edit Curve Properties Editor**, the **Plot Properties Editor**, and the **Default Line Styles Options Editor** (Section 5.5.2).



Edit Curve Properties

Line
○ None
● Custom
Color: blue
Style:
Weight:
Fill: ● None ○ Below ○ Above
Fill Color: blue
Fill Pattern:
Type: ○ Stepped ● Linear
□ Interpolate for missing values

Missing Value Symbols
● None
○ Custom
Style: ◇
Line Color: red
Fill Color: red
Size: 8

Point
○ None
● Custom
Style: ●
Line Color: blue
Fill Color: blue
Size: 8
● Automatic Symbol Drawing
○ Draw Symbols on Data Points
Skip
Offset
○ Draw Symbols at Specified Interval
Interval
Offset

Sample

Figure 5.17 Curve Line and Point Style Editing Interface

The Style tab has three main groups, **Line, Point**, and **Missing Value Symbols** which allow you to customize line and point styles, and a symbol that can be shown for missing values. Beneath the Point group, the **Sample** box provides a preview of the way your line and point choices will look.

To define line styles for curves:

1. In the **Line** group, click **Custom**.
2. Select the color, style, and weight you want for the line.
3. You can display lines with fill above or below, or without fill. Figure 5.18 shows a plot with line fill below, whereas Figure 5.19 shows the same plot without line fill.
4. Select if you want the curve drawn in a stair-stepped style, or linear with a line drawn directly between each point.
5. Select if you want a line drawn to interpolate where there are missing values.

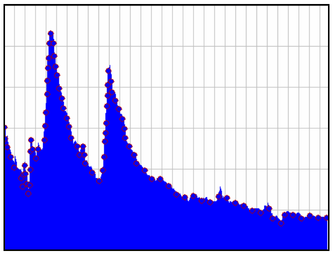


Figure 5.18 Plot with Line Fill

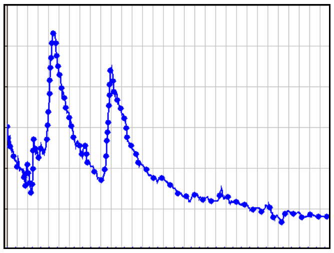


Figure 5.19 Plot without Line Fill

To define point styles for curves:

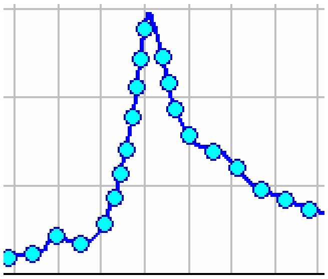


Figure 5.20 Example Line and Fill Colors

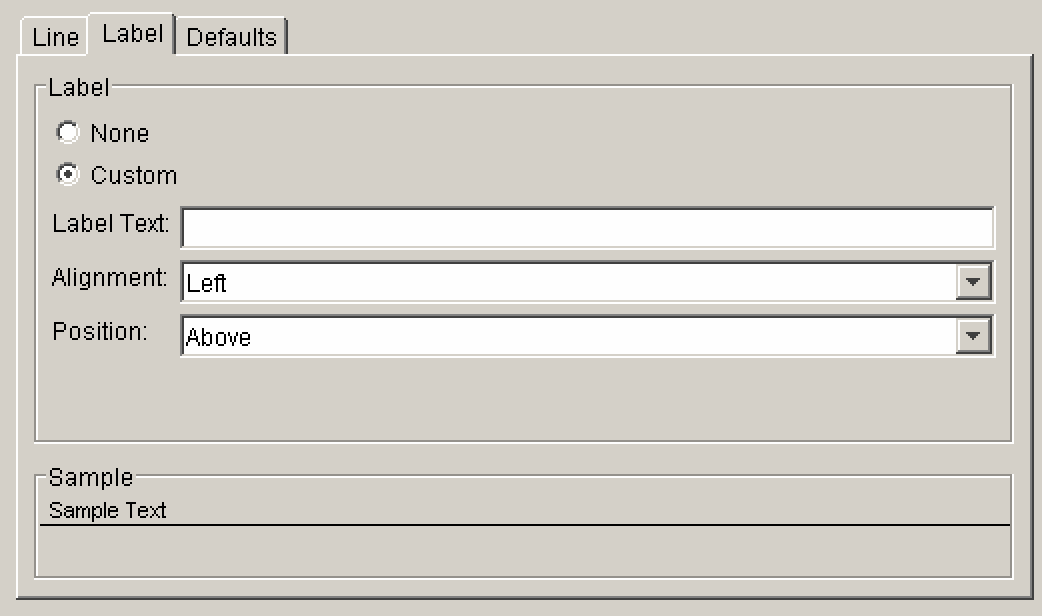
1. In the **Point** group, click **Custom**.
2. Choose the **Style**, **Line Color**, and **Fill Color** you want. The Line Color is the "border" around the point symbol, whereas the Fill Color is the color inside the point symbol. Figure 5.20 shows an example of a dark line color and a light fill color.
3. In the **Size** box, specify the size of the point (in pixels) either by selecting a size from the list or by typing in a number from 1-45.
4. **Automatic Symbol Drawing** allows the plot to compute how to draw the points so they do not overlap. As you zoom in the plot will draw more points because you are increasing the distance between points. This will continue until all points on the curve are drawn.
5. **Draw Symbols on Data Points** allows you to specify how to draw the points so they don't overlap. If you set the **Skip** box to one (1), then it will draw one point, skip the next, then draw the third, etc. The **Offset** box allows you to say how many points on the curve to initially skip before drawing points.

To define missing value symbols for curves:

1. In the **Missing Value Symbol** group, click **Custom**.
2. Choose the **Style**, **Line Color**, and **Fill Color** you want. The Line Color is the "border" around the point symbol, whereas the Fill Color is the color inside the point symbol. Figure 5.20 shows an example of a dark line color and a light fill color.
3. In the **Size** box, specify the size of the point (in pixels) either by selecting a size from the list or by typing in a number from 1-45.
4. This will place the selected symbol on each missing value.

5.5.4 Customizing Curve Labels

Figure 5.21 shows the curve **Label** worksheet, available from the **Curves** tabs of the **Edit Curve Properties Editor**, the **Plot Properties Editor**, and the **Default Line Styles Options Editor** (Section 5.5.2). This worksheet allows you to customize curve labels.



Label

○ None

● Custom

Label Text:

Alignment: Left

Position: Above

Sample

Sample Text

Figure 5.21 Curve Label Editing Interface

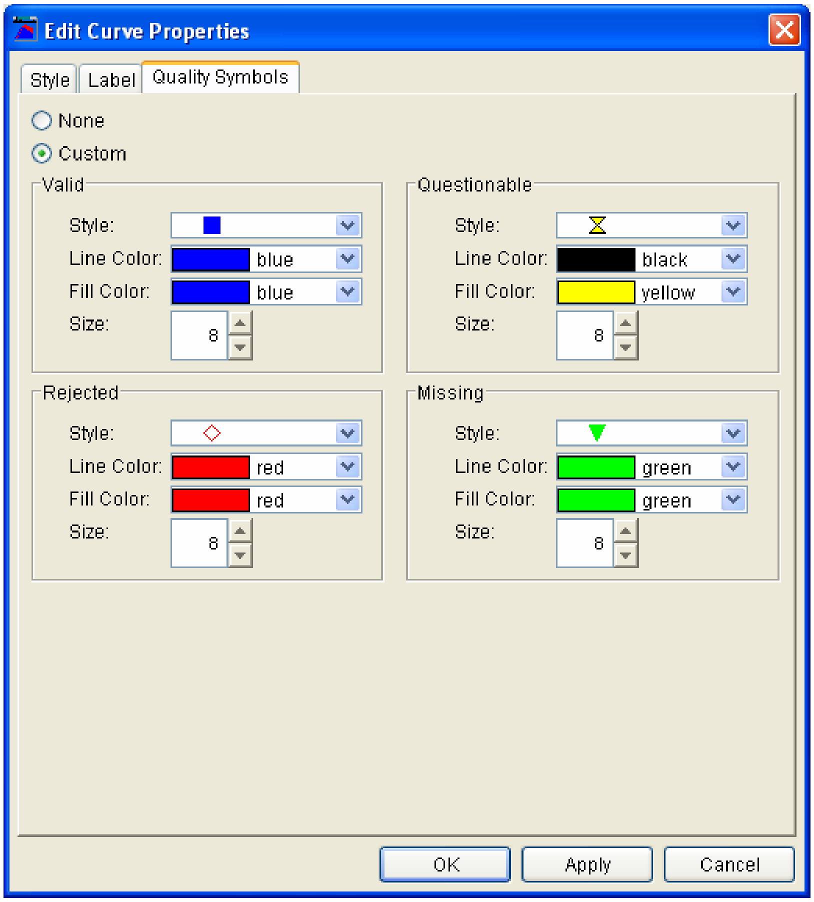
To customize curve labels:

1. Select **Custom**.
2. Enter the text you want to appear in the curve label in the **Label Text** box.
3. In the **Alignment** list, click Left, Center, or Right to select the alignment of the curve label.
4. To set the position of the label, from the **Position** list, click Above, Center, or Below.

The **Sample** box provides a preview of the way your labels will look.

5.5.5 Customizing Curve Quality Symbols

Figure 5.22 shows the curve **Quality Symbols** worksheet, available from the **Curves** tabs of the **Edit Curve Properties Editor**, the **Plot Properties Editor**, and the **Default Line Styles Options Editor** (Section 5.5.2). This worksheet allows you to customize curves for plots that have quality set for their data.



○ None
● Custom

Valid
Style: ■
Line Color: blue
Fill Color: blue
Size: 8

Questionable
Style: ⧗
Line Color: black
Fill Color: yellow
Size: 8

Rejected
Style: ◇
Line Color: red
Fill Color: red
Size: 8

Missing
Style: ▼
Line Color: green
Fill Color: green
Size: 8

Figure 5.22 Curve Quality Symbols Editing Interface

To customize quality symbols:

1. Choose **Custom**.
2. Select a symbol **Style**, **Line Color**, **Fill Color**, and **Size** for each quality of data.

5.6 Customizing Viewport Properties

Viewports are the gridded areas in the plot window that contain plot curves. You can customize the border around the viewport, the background color and pattern, and the appearance of gridlines.

5.6.1 Customizing Viewport Borders and Background

To customize the borders and backgrounds of viewports in an *individual* plot, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Patterns** tab.
  Or
- Right-click in a blank area inside the viewport with the **Pointer Tool**, and then select **Edit Properties** from the shortcut menu (Figure 5.23). When the **Viewport Properties Editor** opens, select the **Patterns** tab.

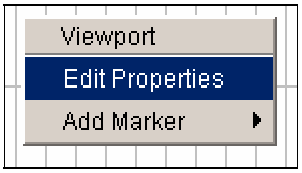
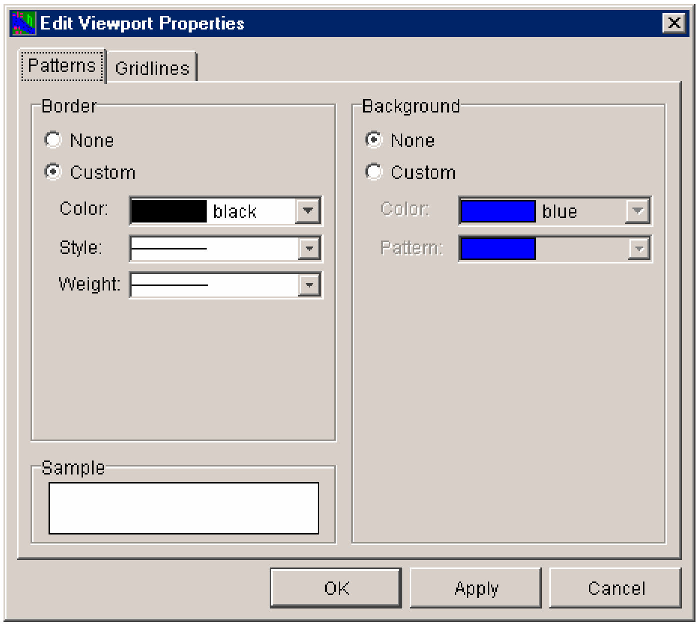


Figure 5.23 Shortcut Menu--Viewport Properties

To specify the default border and background of viewports for *all* of your plots, from the **Edit** menu, click **Default Plot Properties**. When the **Default Plot Properties Editor** opens, choose the **Viewport** tab.

Figure 5.24 shows the **Edit Viewport Properties Editor** with the **Patterns** tab selected. This worksheet, accessed from the shortcut menu, contains the same items as the **Patterns** tab of the **Plot Properties Editor** and the **Viewport** tab of the **Default Plot Properties Editor**.



Edit Viewport Properties

Patterns | Gridlines

Border

○ None
● Custom

Color: black
Style:
Weight:

Sample

Background

● None
○ Custom

Color: blue
Pattern:

Figure 5.24 Viewport Properties Editor--Patterns Tab

The Patterns (or Viewport) tab has two main groups, **Border** and **Background**, which allow you to customize border and background of the viewport, respectively. Beneath the Border group, the **Sample** box provides a preview of the way the viewport border and background will look.

1. Select **Custom** in the **Border** group and choose the **Color**, **Style**, and **Weight** for the border you want to appear around the plot viewport.
2. Select **Custom** in the **Background** group and choose the **Color** and **Pattern** you want for the background.

Click **Apply** to save your changes and continue adjusting the appearance of the border and background. Click **OK** when you are finished.

5.6.2 Customizing Viewport Gridlines

To customize gridlines of viewports in an *individual* plot, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Gridlines** tab.
  Or
- Right-click in a blank area inside the viewport with the **Pointer Tool** , and then select **Edit Properties** from the shortcut menu (Figure 5.25). When the **Viewport Properties Editor** opens, select the **Gridlines** tab.

To specify viewport gridlines for *all* of your plots, from the **Edit** menu, select **Default Plot Properties**. When the **Default Plot Properties Editor** opens, choose the **Grids** tab.

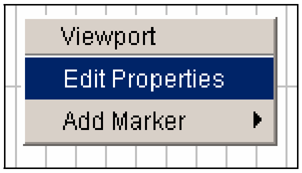


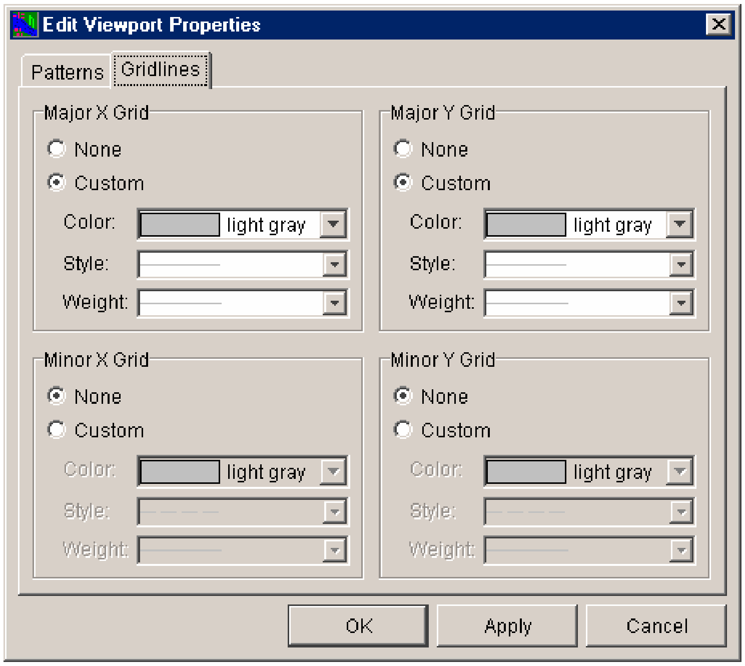
Figure 5.25 Shortcut Menu--Viewport Properties

Figure 5.26 shows the **Viewport Properties Editor** with the **Gridlines** tab selected. This worksheet, accessed from the shortcut menu, contains the same items as the **Gridlines** tab of the **Plot Properties Editor** and the **Grids** tab of the **Default Plot Properties Editor**.

By default, the plot viewport displays gridlines only for the **Major X Grid** and **Major Y Grid**. The default color is light gray. To change the appearance of Major X and Y gridlines, select **Custom** and make your selections for **Color**, **Style**, and **Weight**.

By default, the **Minor X Grid** and **Minor Y Grid** are set to **None** and do not display in the plot viewport. If you want to display gridlines for the Minor X Grid and Minor Y Grid, select **Custom** and make your selections for **Color**, **Style**, and **Weight**.

Click **Apply** to view your changes without closing the editor. Click **OK** when you are finished.



Edit Viewport Properties

Patterns | Gridlines

Major X Grid

○ None
● Custom

Color: light gray
Style:
Weight:

Major Y Grid

○ None
● Custom

Color: light gray
Style:
Weight:

Minor X Grid

● None
○ Custom

Color: light gray
Style:
Weight:

Minor Y Grid

● None
○ Custom

Color: light gray
Style:
Weight:

OK | Apply | Cancel

Figure 5.26 Viewport Properties Editor--Gridlines Tab

5.7 Adding and Customizing Marker Lines

You can add marker lines on your plot's X and Y axes and customize the appearance of these markers.

5.7.1 Adding Markers

To add a marker:

1. Right-click on the location in the plot where you want the marker to appear.
2. From the **Viewport shortcut menu** (Figure 5.27), point to **Add Marker**, and then click either **On X-Axis** or **On Y-Axis**.

The marker will now appear in the plot.

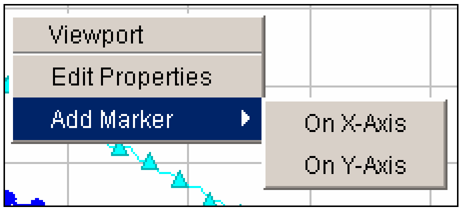


Figure 5.27 Shortcut Menu: Add Marker

5.7.2 Deleting Markers

To delete a marker line in a plot, right click on it with the **Pointer Tool** , and then click **Delete** from the **Marker Line shortcut menu** (Figure 5.28).

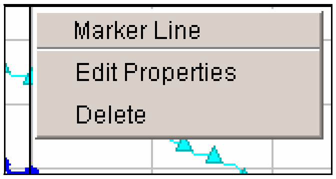


Figure 5.28 Shortcut Menu—Marker Line Properties

5.7.3 Customizing Markers

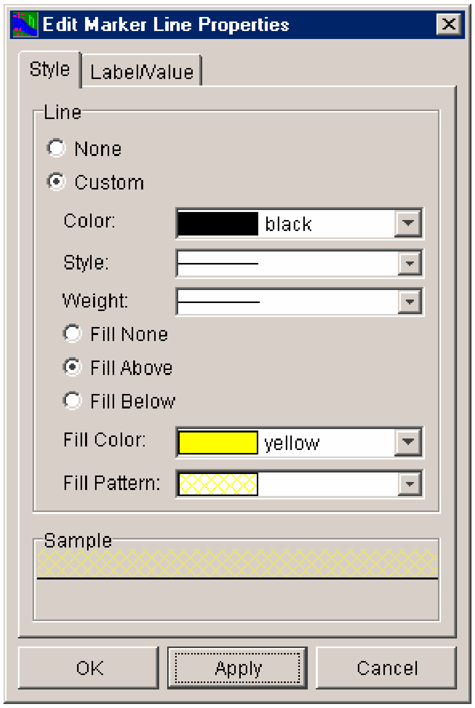
To edit the properties of a marker in an *individual* plot, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Marker** tab. This worksheet is available only if a marker exists in the plot. Choose the marker you want to edit from the **Marker Lines** list.
  Or
- With the **Pointer Tool** , right-click on the marker you want to edit. From the **Marker Line shortcut menu**, select **Edit Properties** (Figure 5.28). The **Edit Marker Line Properties Editor** will open.

To specify the default appearance of markers for *all* of your plots, from the **Edit** menu, click **Default Plot Properties**. When the **Default Plot Properties Editor** opens, choose the **Marker Lines** tab.

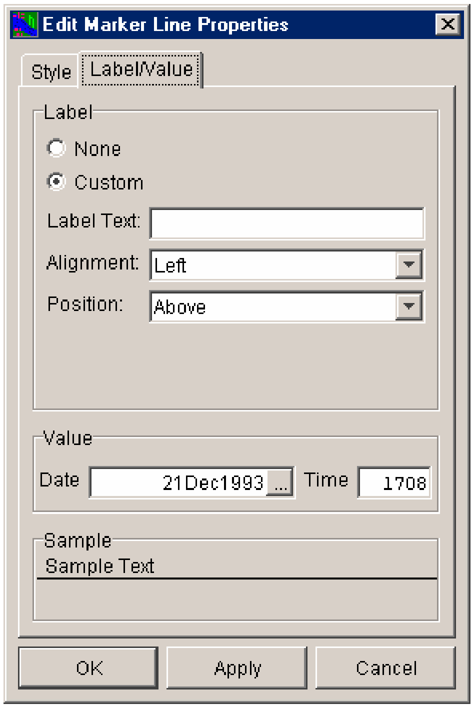
The interfaces for the specialized **Edit Marker Line Editor** and the **Default Plot Properties Editor** are very similar. However, the **Plot Properties Editor** interface differs in two ways. First, the **Marker** worksheet is available only if a marker exists in the current plot. Second, at the top of the editing panel there is a **Marker Lines** list containing all markers that exist in the current plot.

You can edit the line style and label of a marker using two editing tabs. Figure 5.29 shows the **Edit Marker Line Properties Editor** with the **Style** worksheet selected, while Figure 5.30 shows the **Label/Value** worksheet. The Marker Line tabs in the **Plot Properties Editor** and the **Default Plot Properties Editor** share the same two worksheets.



Line
○ None
● Custom
Color: black
Style:
Weight:
○ Fill None
● Fill Above
○ Fill Below
Fill Color: yellow
Fill Pattern:
Sample

Figure 5.29 Marker Line Properties Editor—Style Tab



Label
○ None
● Custom
Label Text:
Alignment: Left
Position: Above
Value
Date 21Dec1993
Time 1708
Sample
Sample Text

Figure 5.30 Marker Line Properties Editor—Label/Value Tab

To edit the line style for markers:

1. Select the **Style** tab.
2. Choose **Custom**.
3. Select the **Color**, **Style**, and **Weight** for the marker line.
4. You can display marker lines with fill **Above** or **Below**, or without fill. Figure 5.31 shows a plot with line Fill Above and a hatched pattern selected.

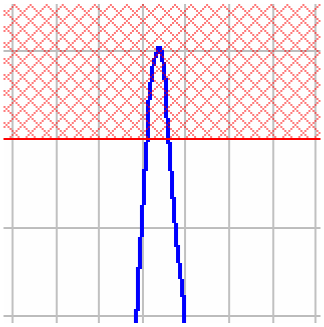


Figure 5.31 Marker Line with Fill Above

The **Sample** box provides a preview of the way the marker line will look.

Click **Apply** to save your changes without closing the editor.

To add a label to a marker line:

1. Select the **Label/Value** tab.
2. Choose **Custom**.
3. In the **Label Text** box, enter the text you want to appear in the label.
4. From the **Alignment** list, select the justification of the label--Left, Center, or Right.
5. To set the position of the label, from the **Position** list, select either Above, Center, or Below.

6. For the **Date** value, enter the date in the format DDMMYYYY (*e.g.*, 21Dec1993). You can also click the ... button to access the **Calendar Tool** (Figure 5.32) to select the date.
7. For the **Time** value, enter the time in 24-hour military format (*e.g.*, for 5:08 pm, enter "1708").

The **Sample** box provides a preview of the way the marker line will look.

Click **Apply** to save and view your changes without closing the editor. Click **OK** when you are finished editing marker line properties.

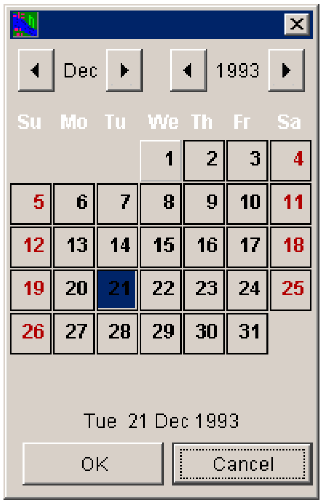


Figure 5.32 Calendar Tool

5.7.4 Editing Callouts

You can add descriptive callouts at specific points along a curve (Figure 5.33). To do this:

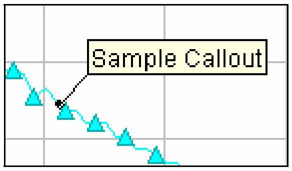


Figure 5.33 Callout

1. Right-click on the location on the curve where you want the callout to appear.
2. From the **shortcut menu** (Figure 5.34), point to **Add Callout**.
3. In the **Add Callout** dialog box (Figure 5.35), enter the text you want to appear in the callout, and then click **OK**.

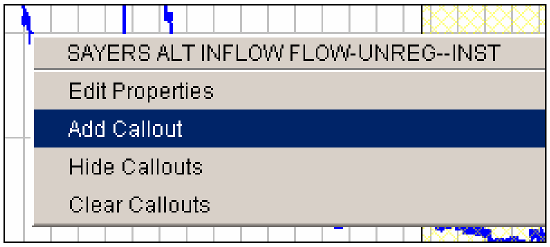


Figure 5.34 Shortcut Menu: Add Callout

To hide all callouts in a plot, right-click on a curve in the plot. From the shortcut menu, click **Hide Callouts**.

Once you have hidden callouts, **Hide Callouts** in the shortcut menu changes to **Show Callouts**, allowing you to return callouts to the plot display.

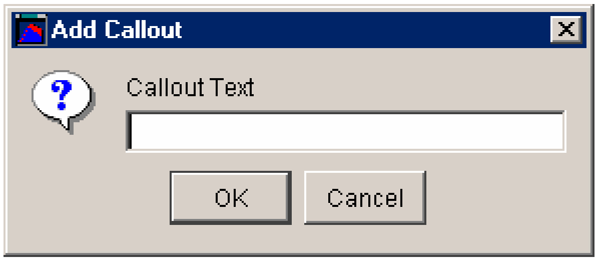


Figure 5.35 Add Callout Dialog Box

To permanently remove all callouts from the plot, right-click on a curve in the plot to access the shortcut menu, then click **Clear Callouts**.

5.8 Customizing Axes

You can choose either a linear or log axis type, specify the axis scale, modify tic marks, and customize axis labels. Probability plots are generated for paired data sets with a type of "PROB", and cannot be changed without changing the data type.

5.8.1 Changing Axis Type

By default, plots display using a linear scale (Linear Axis), in which the axis increases and decreases by x. You can also use the log scale, which allows you to view curves that grow exponentially in a near straight line because the axis increases or decreases by the log (x). For example, you might wish to use the log scale when the axis has evenly-spaced major tics with values of 1,10,100,1000, and so on, such as in a performance history plot showing many years of data.

To change the axis type of an individual plot, right-click on an axis. From the **Axis Tics shortcut menu**, point to **Set Axis Type**, and then select the axis type from the submenu (Figure 5.36). Depending on the axis type, you can click either **Log Axis** or **Linear Axis**.

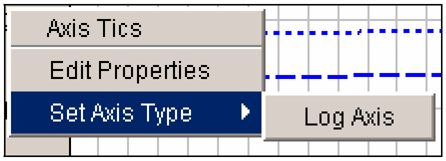


Figure 5.36 Shortcut Menu - Set Axis Type

5.8.2 Specifying Axis Scale

You can specify the axis scale and tic interval for individual plots. To do this, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Axis** tab. From the **Axis** worksheet, select the axis you want to edit from the **Axis** list. Once you have this set, click the **Scale** tab to access the **Scale** worksheet.
  Or
- Right-click on the axis with the **Pointer Tool** . From the **Axis Tics shortcut menu** (Figure 5.37), click **Edit Properties**. When the Edit **Axis Properties Editor** opens, choose the **Scale** tab.

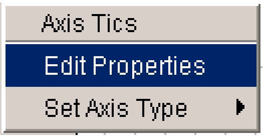
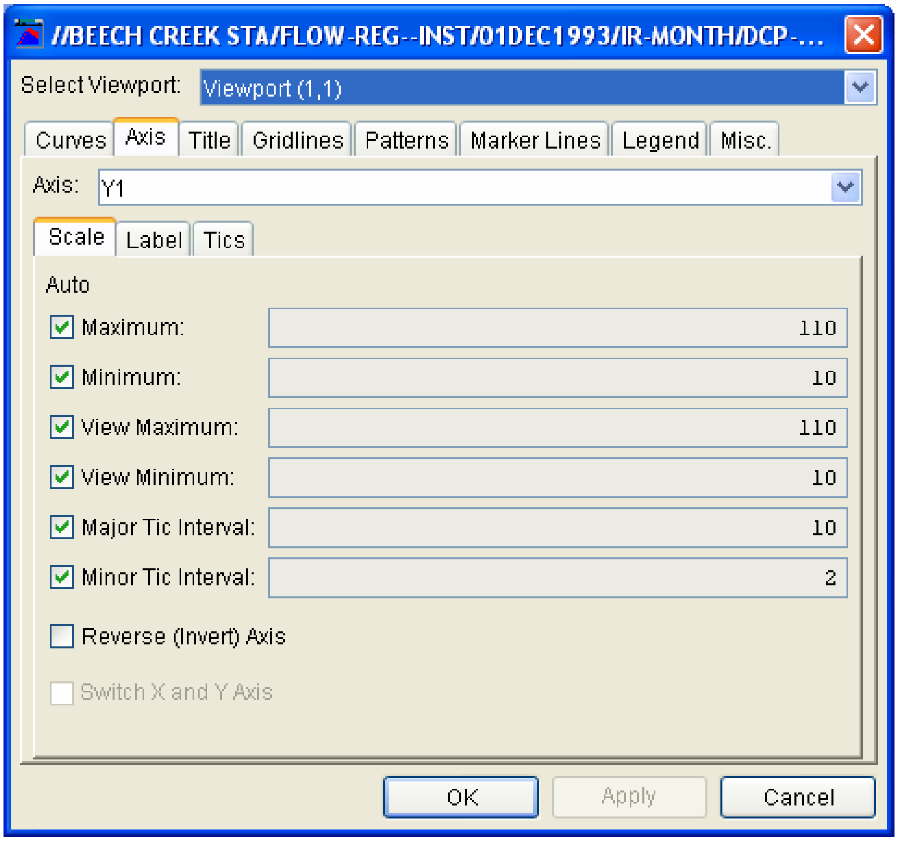


Figure 5.37 Shortcut Menu—Axis Tics

Figure 5.38 shows the **Plot Properties Editor** with the **Axis** tab selected and the **Scale** worksheet open. The **Axis** worksheets of the **Plot Properties Editor** and the **Default Plot Properties Editor** are nearly identical, except for the **Axis** list on the **Plot Properties Editor**. The **Scale** and **Tics** sub-worksheets are available in all three editors.



//BEECH CREEK STA/FLOW-REG--INST/01DEC1993/IR-MONTH/DCP-...

Select Viewport: Viewport (1,1)

Axis: Y1

Auto

☑ Maximum: 110

☑ Minimum: 10

☑ View Maximum: 110

☑ View Minimum: 10

☑ Major Tic Interval: 10

☑ Minor Tic Interval: 2

□ Reverse (Invert) Axis

□ Switch X and Y Axis

Figure 5.38 Plot Properties Editor--Scale Worksheet

With the **Scale** worksheet, you can specify the range of the scale, the amount of the scale that is visible, and the tic intervals.

If the **Auto** box is checked, the plot will automatically select the scale. Otherwise, as you zoom in and out of the plot, its view values change while the minimum and maximum scale range values remain fixed:

- **Maximum:** enter the value of the maximum range of the scale.
- **Minimum:** enter the value of the minimum range of the scale.
- **View Maximum:** enter the maximum visible range of the scale.
- **View Minimum:** enter the minimum visible range of the scale.

Tic intervals are the distances between tics on the axis scale:

- **Major Tic Interval:** specify the distance between each major tic.
- **Minor Tic Interval:** specify a value less than or equal to the major tic value.

You can also choose to reverse the axis and invert the data by selecting Reverse (Invert) Axis. If the data set is paired, you can switch the X and Y axis, so what is plotted on the X axis becomes plotted on the Y axis instead.

Click **Apply** to save and view your changes without closing the editor.

Click **OK** to save your changes and close the editor.

5.8.3 Modifying Tic Marks

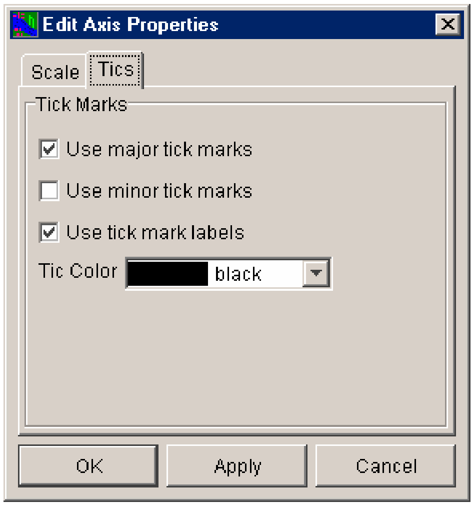
You can modify the color of tic marks, choose whether or not major and minor tic marks display, and specify whether labels display. (See also "Specifying Axis Scale" in Section 5.8.2 for information about modifying tic intervals.)

To modify tic marks in an *individual* plot, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Axis** tab. From the **Axis** worksheet, select the axis you want to edit from the **Axis** list. Once you have this set, choose the **Tics** tab to set tic marks.
  Or
- Right-click on the axis with the **Pointer Tool** . From the **Axis Tics shortcut menu** (Figure 5.37), click **Edit Properties**. When the **Edit Axis Properties Editor** opens, choose the **Tics** tab.

To specify default settings for axis tics in *all* of your plots, click **Default Plot Properties** from the plot Edit menu. When the **Default Plot Properties Editor** opens, choose the **Axis** tab, then the **Tics** tab.

Figure 5.39 shows the **Tics** worksheet of the **Edit Axis Properties Editor**. The **Tics** worksheet is nearly identical on the **Edit Axis Properties Editor**, **Plot Properties Editor**, and **Default Plot Properties Editor**. However, at the top of the **Plot Properties Editor** is an **Axis** list containing all of the axes available for editing in the current plot. In the **Plot Properties Editor**, you must choose an axis to edit before you can make any changes.


Edit Axis Properties

Tick Marks

☑ Use major tick marks

□ Use minor tick marks

☑ Use tick mark labels

Tic Color: black

Figure 5.39 Axis Properties Editor--Tics Tab

By default, plot axes display major tic marks with labels. To turn off these options, click to clear the **Use major tick marks** and **Use tick mark labels** check boxes.

You can also select **Use minor tic marks**.

To change tic color, select the color you want from the **Tic Color** list.

Click **Apply** to save and view changes without closing the editor.

Click **OK** to save your changes and close the editor.

5.8.4 Customizing Axis Labels

You can add borders and backgrounds to axis labels.

To customize axis labels in an *individual* plot, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Axis** tab. On the **Axis** worksheet, select the axis you want to edit from the **Axis** list. Once you have this set, select the **Label** tab to set axis label properties.
  Or
- Right-click on the axis label with the **Pointer Tool** . From the **G2d Label shortcut menu** (Figure 5.40), click **Edit Properties**. The **Edit Label Properties Editor** will open.

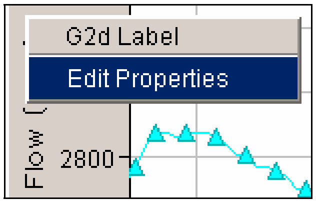
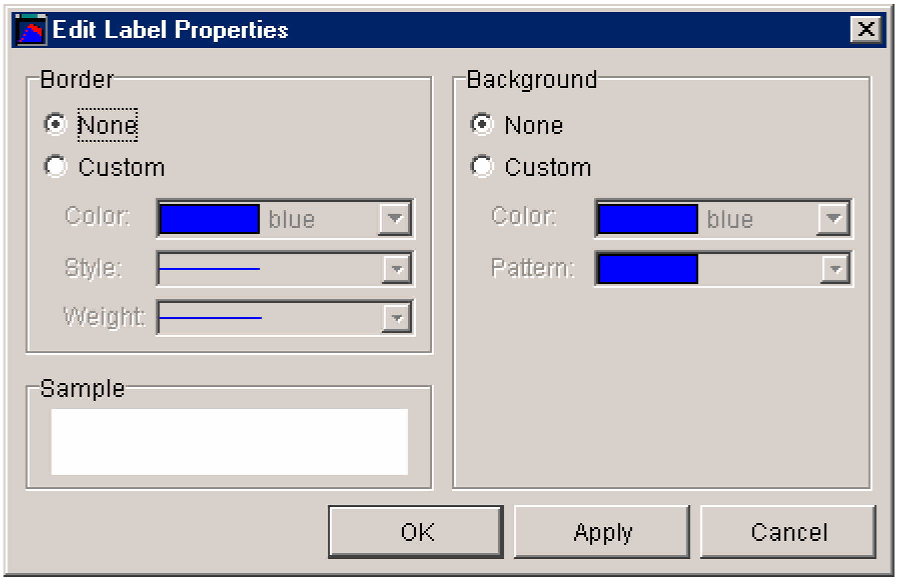


Figure 5.40 G2d Label Shortcut Menu

To specify default settings for axis labels in *all* of your plots, select **Default Plot Properties** from the plot Edit menu. When the **Default Plot Properties Editor** opens, choose the **Axis** tab, then the **Label** tab.

Figure 5.41 shows the **Edit Label Properties Editor**. The same worksheet is available on the **Label** tabs of the **Plot Properties Editor** and **Default Plot Properties Editor**, with one difference. The **Plot Properties Editor** has an **Axis** list containing all of the axes available for editing in the current plot.


Edit Label Properties

Border
● None
○ Custom
Color: blue
Style:
Weight:

Background
● None
○ Custom
Color: blue
Pattern:

Sample

Figure 5.41 Label Properties Editor

The **Label** worksheet has two groups, **Border** and **Background**, which allow you to customize border and background of the label, respectively. Beneath the Border group, the **Sample** box provides a preview of the label.

To add a border around the axis label, choose **Custom** in the **Border** group, and then select the **Color**, **Style**, and **Weight** for the borderline.

To add a background to the axis label, choose **Custom** in the **Background** group, and then select a **Color** and **Pattern**.

Click **Apply** to save and view your changes without closing the editor. Click **OK** when you are finished editing axis label properties.

5.9 Customizing Legends

As Figure 5.42 illustrates, you can add titles to the top of plot legends and add text and graphics to the right and left sides. You can also specify whether the legend appears below or to the right of the plot (refer to Section 5.10.2).

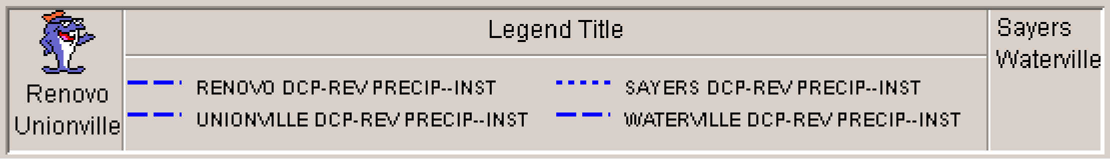

Renovo
Unionville
Legend Title
RENOVO DCP-REV PRECIP--INST
UNIONVILLE DCP-REV PRECIP--INST
SAYERS DCP-REV PRECIP--INST
WATERVILLE DCP-REV PRECIP--INST
Sayers
Waterville

Figure 5.42 Customizing Legends

To add legend labels to an *individual* plot, you can either:

- From the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Legend** tab.
  Or
- Right-click in a blank area inside the legend panel of the plot with the **Pointer Tool** . From the **Legend Panel shortcut menu** (Figure 5.43), click **Edit Properties**. The **Edit Legend Properties Editor** will open.

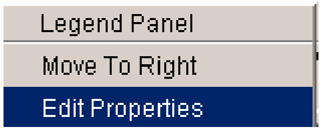
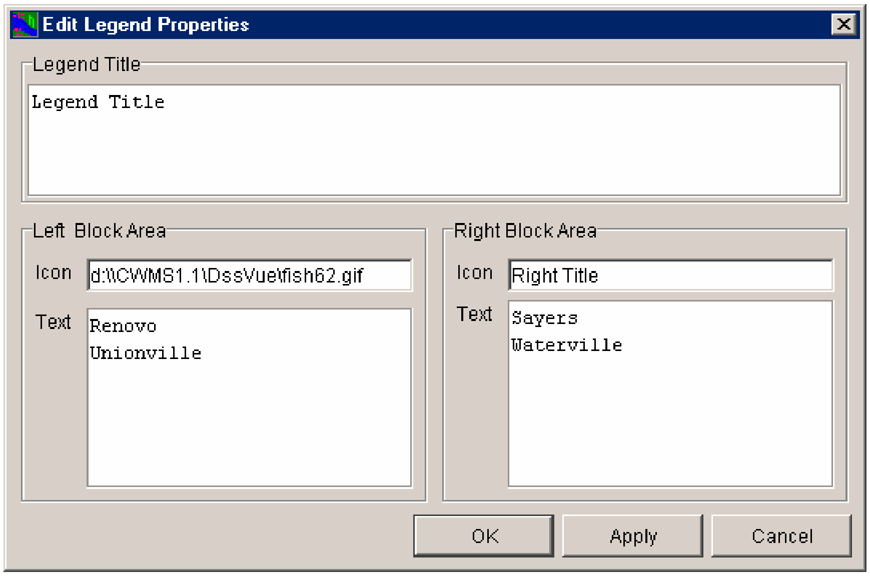


Figure 5.43 Shortcut Menu - Legend Panel

To specify default settings for *all* of your plot legends, from the **Edit** menu, click **Default Plot** Properties. When the **Default Plot Properties** Editor opens, choose the **Legend** tab.

Whether you are using the **Plot Properties Editor**, **Edit Legend Properties Editor**, or **Default Plot Properties Editor**, the worksheet for editing legend titles is the same.

Figure 5.44 shows the **Legend Properties Editor**. This worksheet, accessed from the shortcut menu, contains the same fields as the **Legend** worksheets of both the **Plot Properties Editor** and the **Default Plot Properties Editor**.



Legend Title

Icon d:\\CWMS1.1\DssVue\fish62.gif

Text Renovo
Unionville

Icon Right Title

Text Sayers
Waterville

Figure 5.44 Legend Properties Editor

1. In the **Legend Title** box, enter the title you want to appear along the top of the legend box.
2. In the **Left Block Area** and **Right Block Area** groups, you can specify the **Icon** and **Text** you want to appear on either side of the legend block.
   - If you wish to display icons, type the exact paths and filenames of the icon in the **Icon** field. (The path is the location on your computer.)
   - Enter any text you want to display in the **Text** boxes.
3. Click **Apply** to view your changes without closing the editor. Click **OK** to save your changes and close the editor.

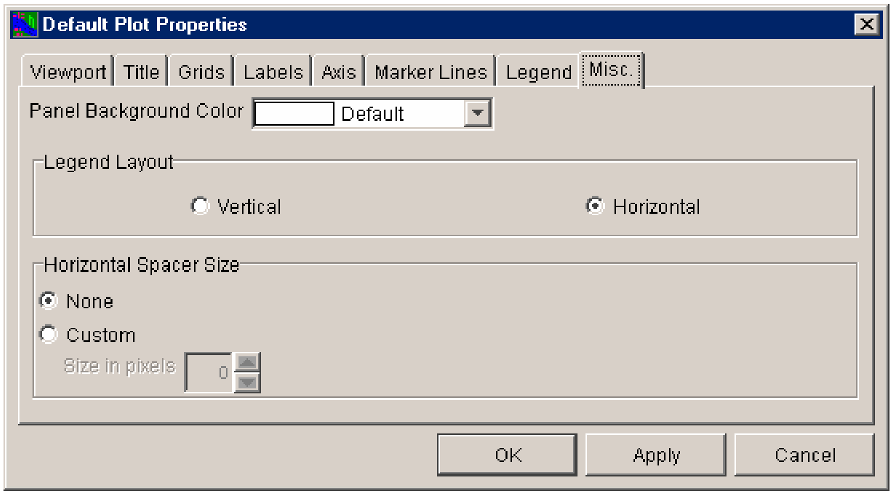
5.10 Customizing Window Panels

You can customize the color and spacing of the window "panel" in which plots are displayed; you can also choose whether the plot legend displays horizontally along the bottom of the plot panel or vertically along the right side.

To customize panel properties of an *individual* plot, from the **Edit** menu, click **Plot Properties**. When the **Plot Properties Editor** opens, select the **Misc.** tab.

To specify panel properties for *all* of your plots, from the **Edit** menu, click **Default Plot Properties**. When the **Default Plot Properties Editor** opens, choose the **Misc.** tab.

Whether you are using the **Plot Properties Editor** or the **Default Plot Properties Editor**, the **Misc**. worksheet is the same. Figure 5.45 shows the **Misc.** worksheet of the **Default Plot Properties Editor**.



Panel Background Color: Default

Legend Layout
○ Vertical ● Horizontal

Horizontal Spacer Size
● None
○ Custom
Size in pixels: 0

Figure 5.45 Default Plot Properties Editor--Misc. Tab

You can also edit the legend layout and spacer size from individual property editors accessed from shortcut menus, as discussed below.

5.10.1 Customizing the Panel Background Color

To customize the panel background color, click the down arrow beside the **Panel Background Color** field to select the color you want. There is no individual property editor available for customizing the panel background color.

5.10.2 Customizing the Legend Layout

If you want the plot legend to display horizontally along the bottom of the plot panel, in the **Legend Layout** group, select **Horizontal**. If you want the plot legend to display vertically along the right side of the plot panel, select **Vertical**.

Another way to adjust the position of the legend for an individual plot is to use the **Legend Panel shortcut menu** (Figure 5.46).

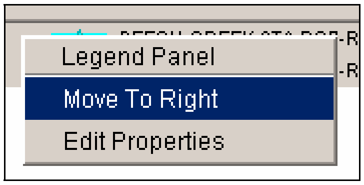


Figure 5.46 Legend Panel Shortcut Menu

To access the Legend Panel shortcut menu, with the pointer tool right-click on a **blank area** inside of the legend panel in the plot window panel. If the legend is currently positioned horizontally, you can click **Move To Right**, which will cause the legend to display vertically. If the legend is currently positioned vertically, you can click **Move to Bottom**, which will cause the legend to display horizontally.

5.10.3 Customizing the Horizontal Spacer Size

In the **Plot Properties Editor** and **Default Plot Properties Editor**, horizontal spacer size refers to the space between viewports in plots with multiple viewports. In single-viewport plots, it is the "margin" space between the right side of the viewport and the edge of the window panel.

To specify the horizontal spacer size using the **Misc.** tab of the **Plot Properties Editor** or **Default Plot Properties Editor**, in the **Horizontal Spacer Size** group, select **Custom**, then either select the pixel size from the list or enter a value less than 50.

You can also use the **Spacer shortcut menu** (Figure 5.47) to customize both horizontal and vertical spacer size for an individual plot.

To access the **Spacer shortcut menu**, right-click on a blank space on the plot panel. Click *between* viewports if you want to adjust vertical spacing between viewports, or *to the right* of a viewport to adjust horizontal spacing.

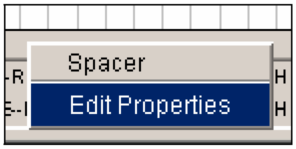
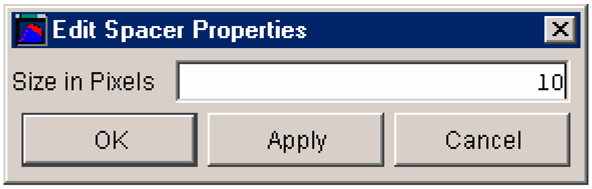


Figure 5.47 Spacer Shortcut Menu

From the **Spacer shortcut menu**, click **Edit Properties**. The **Edit Spacer Properties** dialog box will open (Figure 5.48).


Edit Spacer Properties

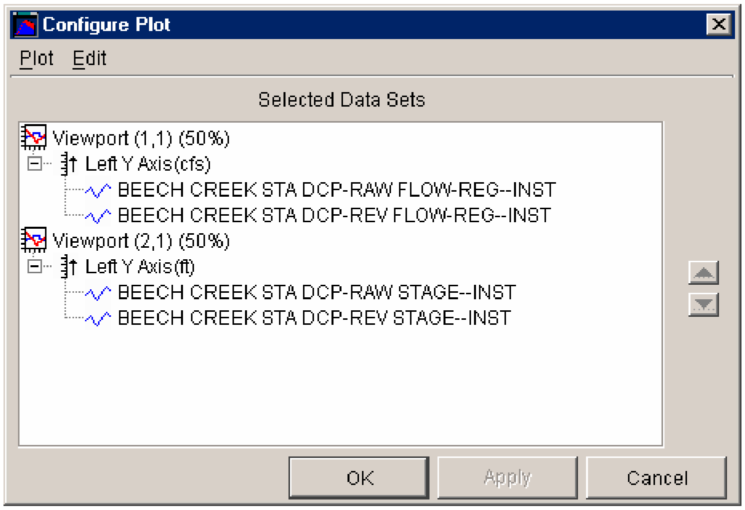
Size in Pixels: 10

Figure 5.48 Edit Spacer Properties Dialog Box

To specify the spacer size, enter a value less than 50.

5.11 Customizing Plot Layout

The **Configure Plot Editor** (Figure 5.49) displays plot components in a "tree" structure and allows you to customize the layout of an individual plot. You can add and remove axes as well as add, remove, arrange the order of, and set the weight of viewports in the plot window panel.



Viewport (1,1) (50%)
Left Y Axis(cfs)
BEECH CREEK STA DCP-RAW FLOW-REG--INST
BEECH CREEK STA DCP-REV FLOW-REG--INST
Viewport (2,1) (50%)
Left Y Axis(ft)
BEECH CREEK STA DCP-RAW STAGE--INST
BEECH CREEK STA DCP-REV STAGE--INST

Figure 5.49 Configure Plot Editor

When you customize the layout of a plot using the **Configure Plot Editor**, your changes apply only to that individual plot unless you export the plot's properties (see Section 5.12).

To access the **Configure Plot Editor**, from the **Edit** menu, select **Configure Plot Layout**.

5.11.1 Adding and Removing Viewports

To add a new viewport to a plot, from the Configure Plot Editor's **Edit** menu click **Add Viewport**. The new viewport will appear at the bottom of the "tree" in the **Selected Data Sets** box.

To remove a viewport, you can either:

- Click on the name of the viewport in the Selected Data Sets "tree." From the **Edit** menu, click **Remove Viewport**.
Or
- Right-click on the viewport's name in the Selected Data Sets "tree." From the shortcut menu (Figure 5.50), click **Remove**.

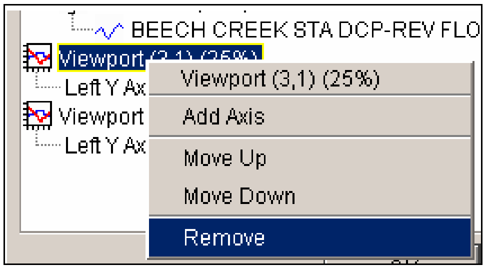
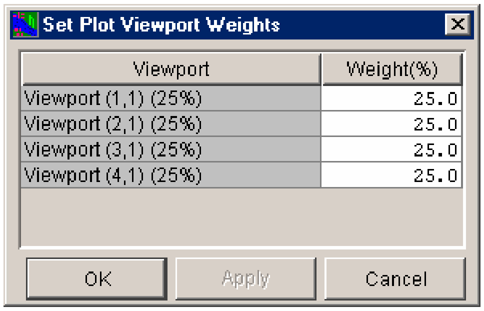


Figure 5.50 Shortcut Menu--Remove Viewport

5.11.2 Setting Viewport Weights

You can customize the relative sizes of viewports in your plots. To do this:

1. From the **Edit** menu of the **Configure Plot Layout Editor**, click **Set Viewport Weights**. The **Set Plot Viewport Weights** dialog box will open.
2. In the **Set Plot Viewport Weights** dialog box (Figure 5.51), you can specify the relative size of each viewport as a percentage value, with all of the weights adding up to 100%. Figure 5.51 shows four viewports of equal weight at 25% each.


Set Plot Viewport Weights

Viewport	Weight(%)
Viewport (1,1) (25%)	25.0
Viewport (2,1) (25%)	25.0
Viewport (3,1) (25%)	25.0
Viewport (4,1) (25%)	25.0

Figure 5.51 Set Plot Viewport Weights Dialog Box

Note that the **Set Plot Viewport Weights** dialog box offers the same right-click shortcut menu commands as are available in all tables. You can cut, copy, and paste data cells; insert, append, and delete rows; and print and export. For more information, refer to the section on tabular data in the **Utilities** chapter.

Click **Apply** to save and view your changes without closing the dialog box. Click **OK** to save your changes and close the dialog box.

5.11.3 Adding and Removing Axes

By default, viewports have a left Y-axis. You can also add a right Y-axis, remove both the left and right Y-axes, and add a new left Y-axis if you have previously removed it. Viewports can have a maximum of two axes, and you cannot remove an axis when a data set is associated with it.

To add an axis to a viewport, you can either:

- Click on the viewport's name in the tree in the Selected Data Sets box. From the **Edit** menu, choose **Add Axis**.
  Or
- Right-click on the viewport's name in the tree in the Selected Data Sets. From the shortcut menu (Figure 5.52), click **Add Axis**.

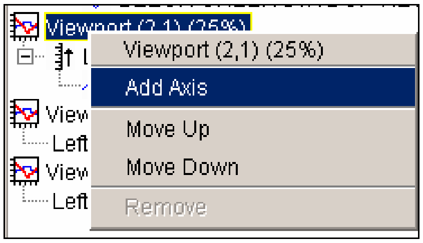


Figure 5.52 Shortcut Menu - Add Axis

The tree now displays another axis for the viewport you selected.

To remove an axis that has no data associated with it, you can either:

- Click on the name of the axis you wish to remove in the tree in the Selected Data Sets box. From the **Edit** menu, click **Remove Axis**.
  Or
- Right-click on the axis you wish to remove in the tree in the Selected Data Sets box. From the shortcut menu, click **Remove**.

The selected axis will no longer display in the tree.

5.11.4 Arranging Viewports and Axes

You can rearrange the vertical order of viewports in a plot window and move axes (with their associated data sets) to different viewports.

To move a viewport or axis,

1. From the tree in the Selected Data Sets box, right-click on the name of a viewport or axis you want to move.
2. From the shortcut menu, click **Move Up** or **Move Down** from the shortcut menu.
3. Either click **OK** or **Apply** for the change to take effect in the plot window.

Note that you cannot move individual data sets in the Configure Plot Layout Editor. You can move only the axis with which a data set is associated.

5.11.5 Reversing Axes (Invert Data)

To reverse the direction of an axis so that the data is inverted, right-click on the name of the axis in the tree. From the shortcut menu, click **Reverse**. Either click **OK** or **Apply** for the change to appear in the plot window.

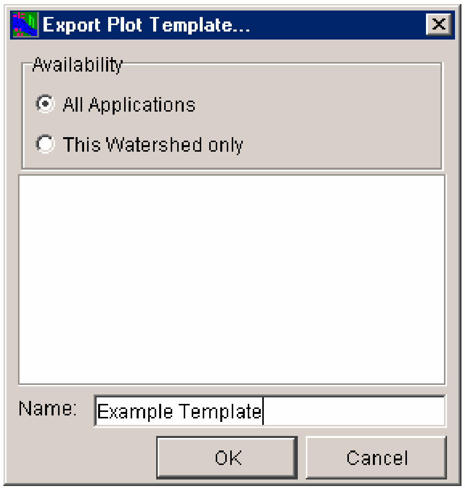
5.12 Exporting and Importing Templates

After you have customized a plot, you can save its settings as a template for use in other plots.

Generally, you will use templates when scripting plots. For example, every day you generate a plot of flow, stage, and precipitation via a script, and then apply a template that has all of the correct formatting, such as viewport placement, size, line colors, and fills. For more information about using templates with scripts, refer to the chapter on **Scripting**.

To create a template from a plot:

1. From the **File** menu, click **Export Properties**.
2. From the **Export Plot Template** dialog box (Figure 5.53), specify whether you want the template to be available for **All Applications** or **This Watershed only**, and then give the template a **Name**.
3. Click **OK** to save the template and close the dialog box.



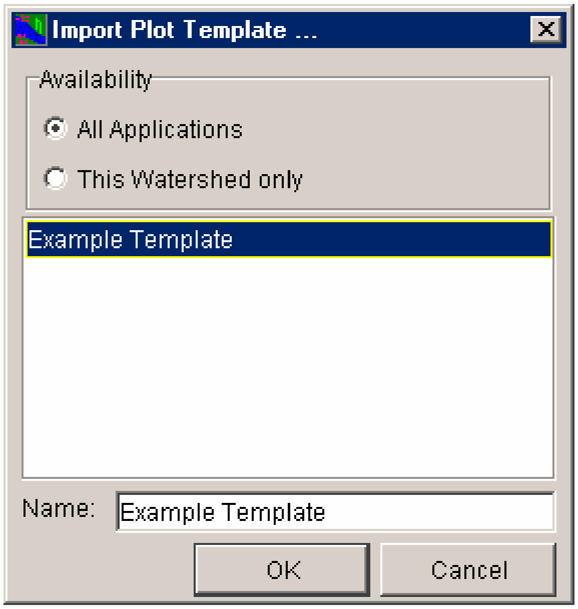
● All Applications
○ This Watershed only

Name: Example Template

Figure 5.53 Export Plot Template Dialog Box

To apply (import) a template you have created to another plot currently open:

1. From the **File** menu, click **Import Properties**.
2. In the **Import Plot Template** dialog box (Figure 5.54), specify whether you want to use a template available for **All Applications** or for **This Watershed only**.
3. A list of available templates will display. Choose the template you want by clicking on its name.
4. When you select a template, its name will display in the **Name** field.
5. Click **OK** to apply the template to the current plot and close the dialog box.



● All Applications
○ This Watershed only

Example Template

Name: Example Template

Figure 5.54 Import Plot Template Dialog Box

5.13 Additional Viewing Options for Plots

The **File** menu of plots (Figure 5.55) contains several commands that allow you to view plot data in tabular form, save plots, and copy and paste plots into other applications such as Microsoft Word, Excel, and Visio.

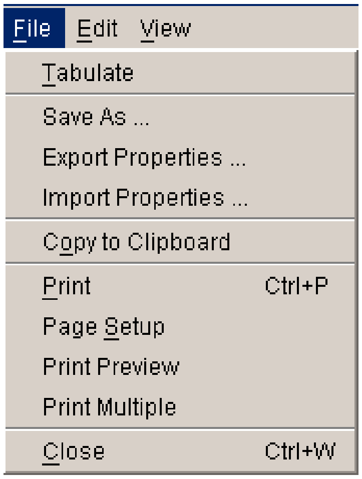
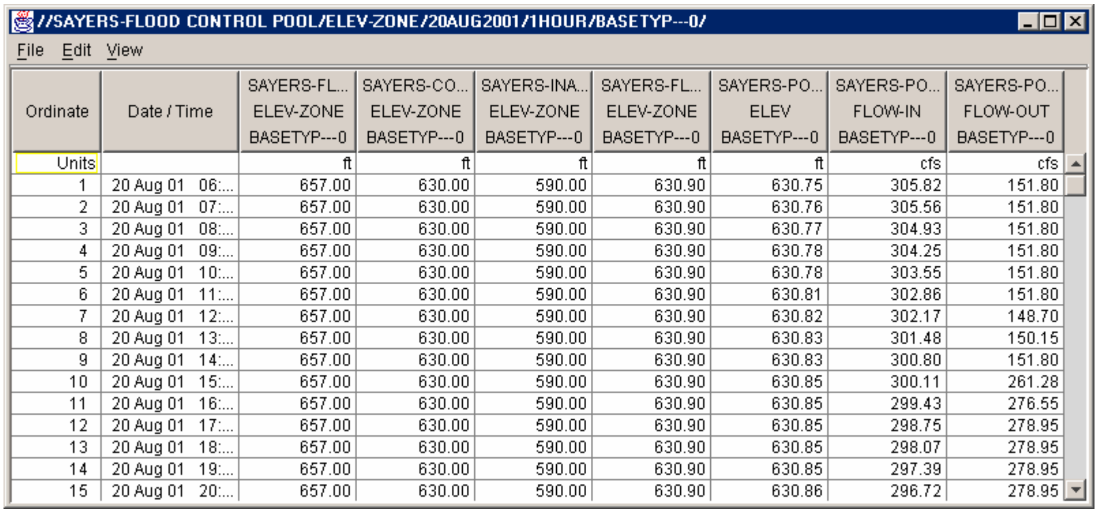


Figure 5.55 File Menu

5.13.1 Viewing Data in Tabular Form

To view plot data in tabular form, from the **File** menu, click **Tabulate**. A dialog box will open displaying the data in tabular form (Figure 5.56).


//SAYERS-FLOOD CONTROL POOL/ELEV-ZONE/20AUG2001/1HOUR/BASETYP---0/
File
Edit
View

Ordinate	Date / Time	SAYERS-FL... ELEV-ZONE BASETYP---0	SAYERS-CO... ELEV-ZONE BASETYP---0	SAYERS-INA... ELEV-ZONE BASETYP---0	SAYERS-FL... ELEV-ZONE BASETYP---0	SAYERS-PO... ELEV BASETYP---0	SAYERS-PO... FLOW-IN BASETYP---0	SAYERS-PO... FLOW-OUT BASETYP---0
Units		ft	ft	ft	ft	ft	cfs	cfs
1	20 Aug 01 06:...	657.00	630.00	590.00	630.90	630.75	305.82	151.80
2	20 Aug 01 07:...	657.00	630.00	590.00	630.90	630.76	305.56	151.80
3	20 Aug 01 08:...	657.00	630.00	590.00	630.90	630.77	304.93	151.80
4	20 Aug 01 09:...	657.00	630.00	590.00	630.90	630.78	304.25	151.80
5	20 Aug 01 10:...	657.00	630.00	590.00	630.90	630.78	303.55	151.80
6	20 Aug 01 11:...	657.00	630.00	590.00	630.90	630.81	302.86	151.80
7	20 Aug 01 12:...	657.00	630.00	590.00	630.90	630.82	302.17	148.70
8	20 Aug 01 13:...	657.00	630.00	590.00	630.90	630.83	301.48	150.15
9	20 Aug 01 14:...	657.00	630.00	590.00	630.90	630.83	300.80	151.80
10	20 Aug 01 15:...	657.00	630.00	590.00	630.90	630.85	300.11	261.28
11	20 Aug 01 16:...	657.00	630.00	590.00	630.90	630.85	299.43	276.55
12	20 Aug 01 17:...	657.00	630.00	590.00	630.90	630.85	298.75	278.95
13	20 Aug 01 18:...	657.00	630.00	590.00	630.90	630.85	298.07	278.95
14	20 Aug 01 19:...	657.00	630.00	590.00	630.90	630.85	297.39	278.95
15	20 Aug 01 20:...	657.00	630.00	590.00	630.90	630.86	296.72	278.95

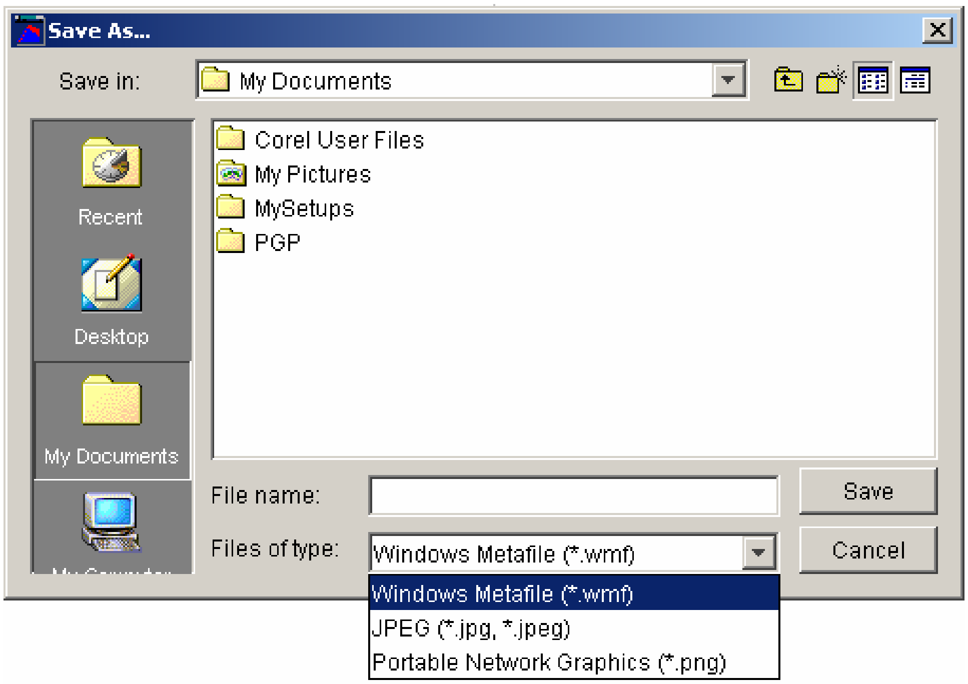
Figure 5.56 Data in Tabular Form

For more information about tables, refer to the chapter on **Utilities**.

5.13.2 Saving Plots

You can save a plot as an image (JPEG), a Windows Metafile (*.wmf), as Portable Network Graphics (*.png), or other formats.

To do this, from the **File** menu, click **Save As**. From the **Save** dialog box (Figure 5.57), select the location where you want to save the plot, enter a filename in the **File name** box, and select the file type you want from the **Files of type** list, then click **Save**.



Save in: My Documents

Files of type: Windows Metafile (*.wmf)

Figure 5.57 Save Dialog Box

5.13.3 Copying Plots to the Clipboard

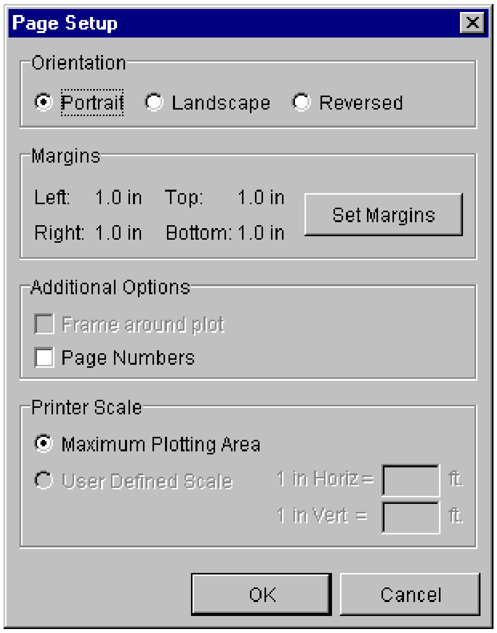
Use **Copy to Clipboard** from the **File** menu to copy a plot to the clipboard. You can then paste the plot as an image into another application such as Microsoft Word, Excel, or Visio.

5.14 Printing Plots

The **Print** command, available from the **File** menu of plots, opens a standard, Windows-style print dialog box.

From the **File** menu, click **Page Setup**. The **Page Setup** dialog box will open (Figure 5.58). Here you can set the page **Orientation**, **Page Margins**, **Page Numbers**, and **Printer Scale**.

Set Margins opens the **Printer Margins** dialog box (Figure 5.59).



Page Setup

Orientation

● Portrait ○ Landscape ○ Reversed

Margins

Left: 1.0 in Top: 1.0 in

Right: 1.0 in Bottom: 1.0 in

Additional Options

□ Frame around plot

□ Page Numbers

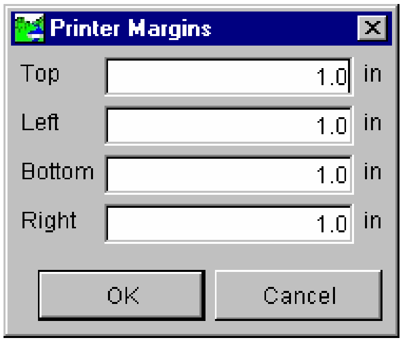
Printer Scale

● Maximum Plotting Area

○ User Defined Scale 1 in Horiz = ft.

1 in Vert = ft.

Figure 5.58 Page Setup Dialog Box



Printer Margins

Top 1.0 in

Left 1.0 in

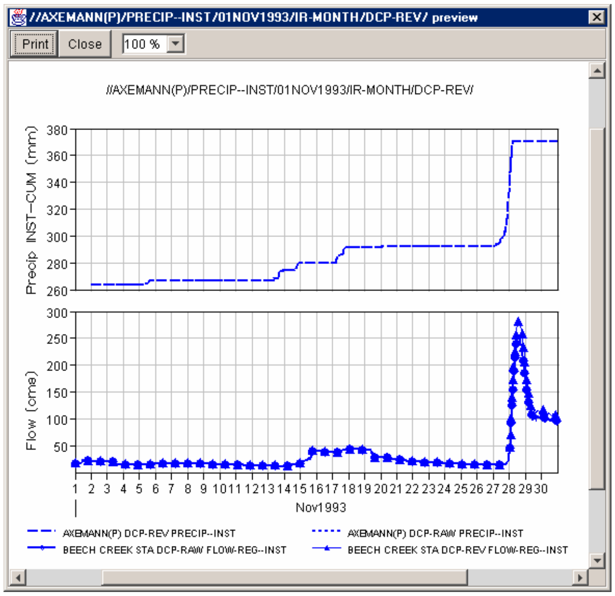
Bottom 1.0 in

Right 1.0 in

Figure 5.59 Printer Margins Dialog Box

To view the plot as it will be printed, from the **File** menu, click **Print Preview**. Figure 5.60 shows an example.

Finally, the **Print Multiple** command allows you to print several plots on one page. The **Print Multiple** dialog box (Figure 5.61) shows all of the currently open plots in the **Available Plots** box. To select plots for printing, double-click on the plots in the **Available Plots** box and the selected plots will move to the **Selected Plots** box.

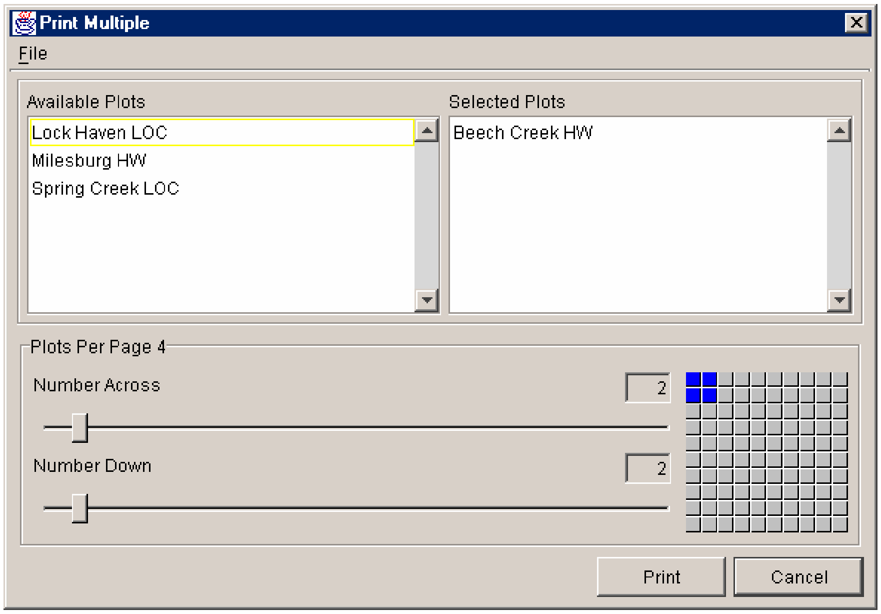


//AXEMANN(P)/PRECIP--INST/01NOV1993/IR-MONTH/DCP-REV/ preview

Print
Close
100 %
//AXEMANN(P)/PRECIP--INST/01NOV1993/IR-MONTH/DCP-REV/
Precip INST–CUM (mm)
380
360
340
320
300
280
260
Flow (cms)
300
250
200
150
100
50
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Nov1993
AXEMANN(P) DCP-REV PRECIP--INST
AXEMANN(P) DCP-RAW PRECIP--INST
BEECH CREEK STA DCP-RAW FLOW-REG--INST
BEECH CREEK STA DCP-REV FLOW-REG--INST

Figure 5.60 Example Print Preview of a Plot

Next, use the slider bars to specify the number of plots you wish to appear horizontally and vertically on the page. The grid to the right of the sliders reflects your choices.



Available Plots

Lock Haven LOC
Milesburg HW
Spring Creek LOC

Selected Plots

Beech Creek HW

Plots Per Page 4

Number Across 2

Number Down 2

Figure 5.61 Print Multiple Dialog Box

Figure 5.62 shows a preview of the plots set up above. From the **Print Multiple** dialog box, you can also use the **Page Setup** and **Print Preview** commands. The two commands are available from the **File** menu of the **Print Multiple** dialog box.

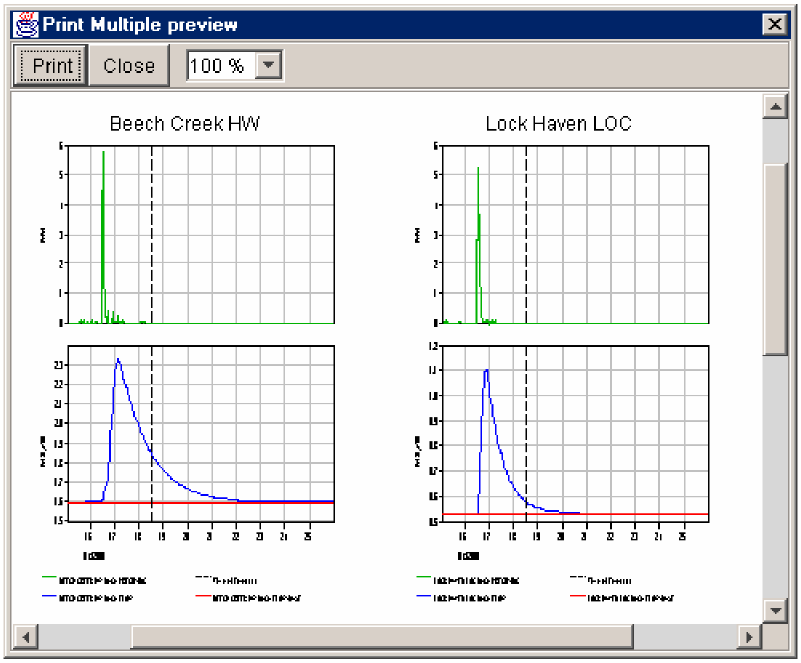


Figure 5.62 Print Multiple Preview Dialog Box (Example)

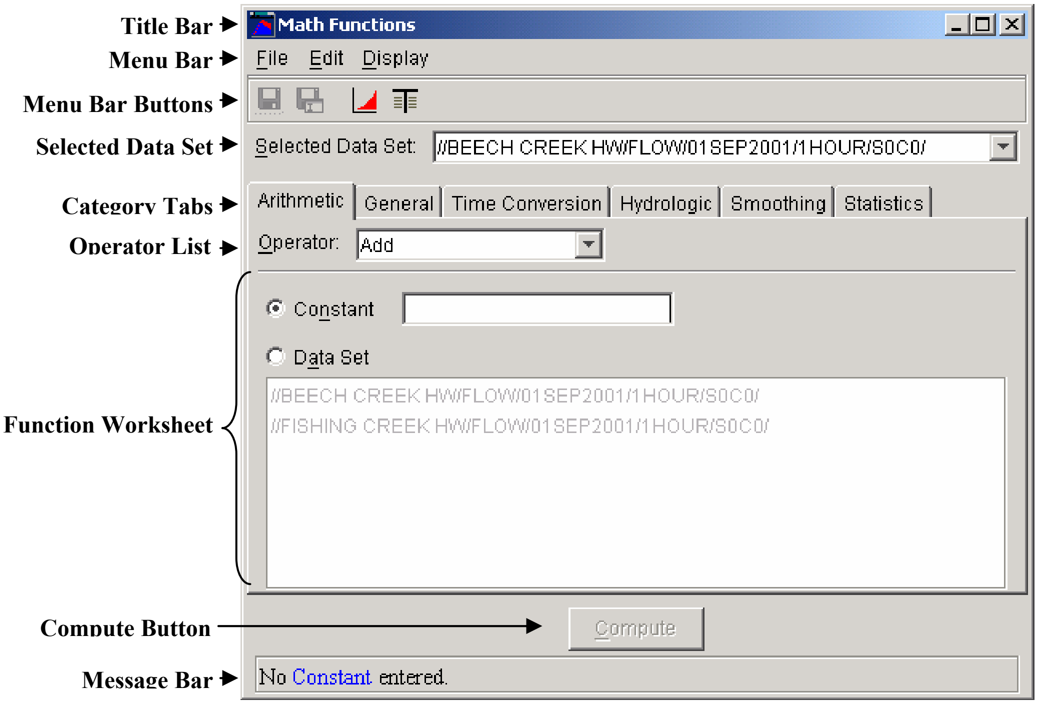
Chapter 6

Math Functions

The Math functions are available from the **Utilities** menu or Math Functions Toolbar button and are organized into six categories: Arithmetic, General, Time Conversion, Hydrologic, Smoothing and Statistics. Each function category is a tab on the **Math Functions** screen (Figure 6.1). This chapter describes math functions according to these categories.

6.1 The Math Functions Screen

Figure 6.1 shows the basic appearance of the **Math Functions** screen. The HEC-DSSVue Math Functions screen enables the mathematical manipulation of time series and paired data selected in the HEC-DSSVue Data Selection List window.



Selected Data Set: //BEECH CREEK HW/FLOW/01SEP2001/1HOUR/S0C0/

Operator: Add

○ Constant

○ Data Set

//BEECH CREEK HW/FLOW/01SEP2001/1HOUR/S0C0/
//FISHING CREEK HW/FLOW/01SEP2001/1HOUR/S0C0/

No Constant entered.

Figure 6.1 Math Functions Screen

6.1.1 Menu Bar

Menu options in the Math Functions screen allow you to save or rename data computed with the Math functions, and to display the computed and original data in plots and tables. The Math Functions menus are as follows:

File **File** menu commands are **Save**, **Save As**, and **Close**.

Edit The **Edit** menu contains the **Restore Original Data** command.

Display Use the **Display** menu to open plots and tables with the **Plot** and **Tabulate** commands, and set options for data to display with the **Original Data with computed** and **All Data Sets** commands.

6.1.2 Menu Bar Buttons

Menu bar buttons provide shortcuts to frequently used Menu commands:

💾 Saves computed data to a HEC-DSS file (same as **Save** in the **File** menu).

💾 Saves and Renames computed data to a HEC-DSS file (same as **Save As** in the **File** menu).

📈 Displays data as a Plot (same as **Plot** in the **Display** menu)

▤ Displays data in Tabular form (same as **Tabulate** in the **Display** menu).

6.1.3 Other Math Features of the Math Functions Screen

The **Operator** and **Selected Data Set** items appear for all function types.

Use the **Category Tabs** to access each of the six categories of math functions.

To select a function, use the **Operator** list.

Use the **Selected Data Set** list to choose a data set to apply each function. This list contains the names of data sets you have chosen in the HEC-DSSVue Data Selection List window.

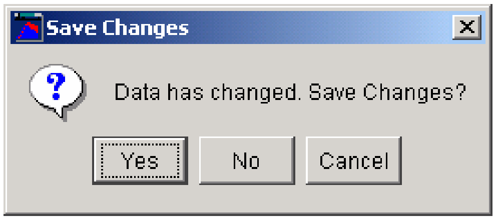
Use the **Compute** button located near the bottom of the screen to apply a function to selected data sets. If data on the function screen is incomplete, the Compute button is unavailable. A message appears in the **Message Bar** at the bottom of the screen indicating which box is incomplete. In Figure 6.1, for example, the message indicates that no constant has been entered.

6.2 Managing Data

Most functions modify the values in the selected data. However, the Math Functions screen retains a copy of the original data, which you may use for comparison plotting with the computed result or for "undoing" the compute action. Other functions, such as "Merge Time Series," may generate a new data set, which is appended to the data set list in the **Selected Data Set** list.

Once a data set has been modified or generated by a function compute, you can save the data to the file, plot it, or tabulate it using the menu options or menu bar buttons located at the top of the Math Functions screen. A computed data set is not automatically saved to the file until you explicitly save the operation.

When you exit the Math Functions screen, if one or more data sets have been modified but not saved, you will be prompted with the **Save Changes** message box shown in Figure 6.2. Save the changes by clicking **Yes**.


Save Changes

Data has changed. Save Changes?

Figure 6.2 Save Changes Dialog Box

In plots and tables, the F-Part of the pathname is identified as "MODIFIED-." This is not retained when the data set is saved to the file and saving the resulting data will overwrite records in the file. In Figure 6.3, the "BEECH CK STATION S0C0T0" time series was added to the "BEECH CREEK HW S0C0" time series. The resulting data set is "BEECH CK STATION MODIFIED-S0C0T0.

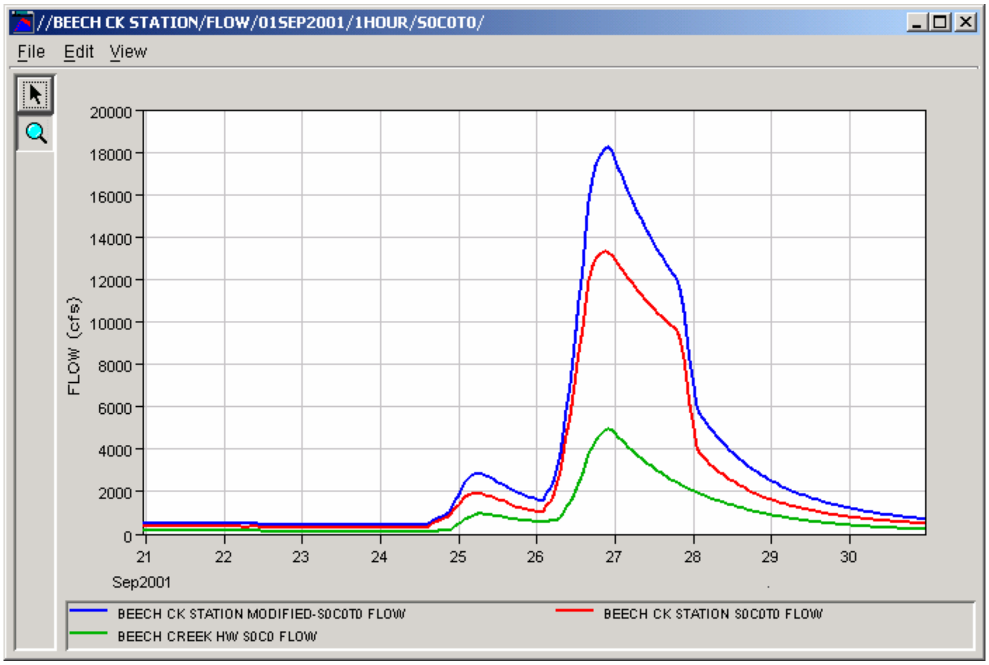


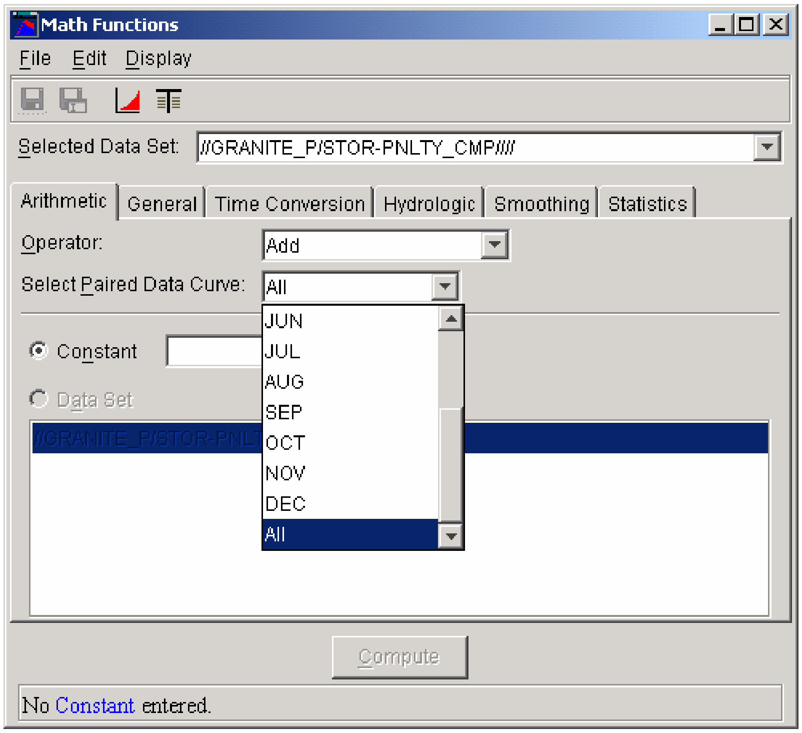
Figure 6.3 Plot of Time Series Data from the "Add" Function

6.2.1 Selecting Paired Data Curves for Function Operations

Typically paired data has a set of x-values and a corresponding set of y-values (e.g. a stage-flow rating table). However, a paired data set may have multiple sets of y-values, or curves, which share the same x-ordinates (e.g. frequency-damage curves for multiple categories). Math functions that operate on paired data sets allow the user to select one or all of the paired data curves for the function application. For example, with the Add function, a number may be added to y-values in one curve or to y-values in all curves.

Figure 6.4 shows the appearance of the Math Functions screen when a paired data set is selected for an "Add" operation.

The paired data curve to apply the add function is picked using the **Select Paired Data Curve** list. The list is filled with the paired data curve labels. In this case, there is a curve for each month of the year. The "All" selection in the list refers to the option to pick all curves for the function operation. If the paired data curves are not explicitly labeled in the paired data set, the curve list is then filled with the curve numbers. The **Select Paired Data Curve** list only appears if the selected data set is paired data.



Selected Data Set: //GRANITE_P/STOR-PNLTY_CMP////

Operator: Add

Select Paired Data Curve: All

● Constant

○ Data Set

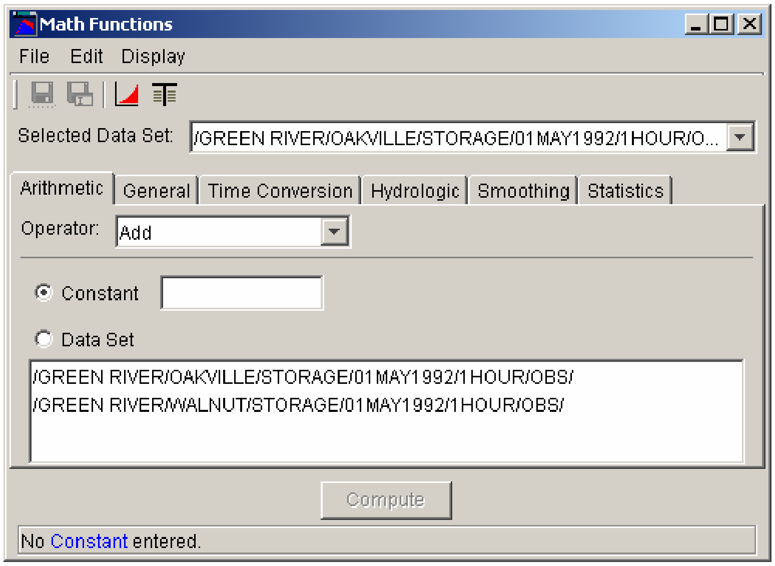
//GRANITE_P/STOR-PNLT

No Constant entered.

Figure 6.4 Selection of Paired Data Curve.

6.3 Arithmetic Functions

The **Arithmetic** tab (Figure 6.5) contains the functions Add, Subtract, Multiply, Divide, Exponentiation, Absolute Value, Square Root, Log, Log Base 10, Sine, Cosine, Tangent, Inverse (1/X), Accumulation, Successive Differences, and Time Derivative.



Selected Data Set: /GREEN RIVER/OAKVILLE/STORAGE/01MAY1992/1HOUR/O...

Operator: Add

● Constant

○ Data Set

/GREEN RIVER/OAKVILLE/STORAGE/01MAY1992/1HOUR/OBS/
/GREEN RIVER/WALNUT/STORAGE/01MAY1992/1HOUR/OBS/

No Constant entered.

Figure 6.5 Math Functions--Arithmetic Tab

6.3.1 Add

1. data set at a time.
2. Click **Constant** and enter the value to multiply by in the box to the right.
3. Click **Compute**.

To multiply a second or more data sets to the selected data set (time series data sets only):

1. From the **Selected Data Set** list, select a data set to apply the function. This data set will contain the result of the multiply operation.
2. Click **Data Set**.
3. From the **Data Set** list, select the data sets to be multiplied. If you include the data set selected in the top **Selected Data Set**, it will be multiplied by itself.
4. Click **Compute**.

6.3.2 Divide

The **Divide** function divides all valid values in a time series or paired data set by a number, or divides the values in a data set by the values in one or more data sets of the same type. For time series data, missing values are kept as missing.

When you are dividing data sets, times in the data sets need not match exactly. However, only values with coincident times will be divided. Times in the primary time series data set that cannot be matched to times in the other data set result in missing values for those times. Values in the data set(s) selected from the lower list form the divisor(s). Values in the resultant data set are set to missing if there is a zero divisor. Missing values in the primary data set are kept as missing Data sets may be regular or irregular interval time series.

Presently the Divide function does not allow the division of paired data by another data set.

To divide all values in the selected data set(s) by a number:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Divide** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details). Constants can be divided by only one paired data set at a time
4. Click **Constant** and enter the value to divide by in the box to the right.
5. Click **Compute**.

To divide the selected data set by one or more data sets (time series only):

1. From the **Selected Data Set** list, select a data set to apply the function. This data set will contain the result of the divide operation.
2. Click **Data Set**.
3. From the **Data Set** list, select the divisor data set(s). . If you include the data set selected in the top **Selected Data Set**, it will be divided by itself.
4. Click **Compute**.

6.3.3 Exponentiation

The **Exponentiation** function raises values in a time series or paired data set to a user specified power, or exponent. For time series data, missing values are kept as missing.

For exponentiation of the values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Exponentiation** operator.

2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. In the **Power** box, enter the value for the power, or exponent,.
5. Click **Compute**.

6.3.4 Absolute Value

The **Absolute Value** function computes the absolute value of values in a time series or paired data set. For time series data, missing values are kept as missing.

To compute the absolute value of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Absolute Value** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. Click **Compute**.

6.3.5 Square Root

The **Square Root** function computes the square root of valid values in a time series or paired data set. If a value is less than 0.0, the value is set to missing in the resultant data set. For time series data, if the original value is missing, the value remains missing in the resultant data set.

To compute the square root of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Square Root** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section

6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).

4. Click **Compute**.

6.3.6 Log

The **Log** function computes the natural logarithm (log base "e") of valid values in a time series or paired data set. If a value is less than or equal to 0.0, the value is set to missing in the resultant data set. For time series data, if the original value is missing, the value remains missing in the resultant data set.

To compute the natural logarithm of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Log** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1, "Selecting Paired Data Curves for Function Operations," for more details).
4. Click **Compute**.

6.3.7 Log Base 10

The **Log Base 10** function computes the log base 10 of valid values in a time series or paired data set. If a value is less than or equal to 0.0, the value is set to missing in the resultant data set. For time series data, if the original value is missing, the value remains missing in the resultant data set.

To compute the log base 10 of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Log Base 10** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. Click **Compute**.

6.3.8 Sine

The **Sine** function computes the sine of valid values in a time series or paired data set. The resulting data set will be in radians. For time series data, missing values are kept as missing.

To compute the sine of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Sine** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. Click **Compute**.

6.3.9 Cosine

The **Cosine** function computes the cosine of valid values in a time series or paired data set. The resulting data set will be in radians. For time series data, missing values are kept as missing.

To compute the cosine of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Cosine** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. Click **Compute**.

6.3.10 Tangent

The **Tangent** function computes the tangent of valid values in a time series or paired data set. The resulting data set will be in radians. If the cosine of a value is 0.0, the value is set to missing in the resultant data set. For time series data, if the original value is missing, the value remains missing in the resultant data set.

To compute the tangent of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Tangent** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. Click **Compute**.

6.3.11 Inverse

The **Inverse** function computes a new value from 1 divided by the value (1/x) in a time series or paired data set. If a value is equal to 0.0, the value is set to missing in the resultant data set. For time series data, if the original value is missing, the value remains missing in the resultant data set.

To compute the inverse (1/x) of values:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Inverse** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. If the data set is paired data, use the **Select Paired Data Curve** list to select a single paired data curve or all curves for the operation (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
4. Click **Compute**.

6.3.12 Accumulation

The **Accumulation** function computes a running accumulation of values for a regular or irregular interval time series data set. For a missing value in the time series data, the value in the accumulation time series remains constant (i.e., missing values treated as zero).

To compute the running accumulation of values for time series data sets:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Accumulation** operator.

2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. Click **Compute**.

6.3.13 Successive Differences

The **Successive Differences** function computes the difference between successive values in a regular or irregular interval time series data set. The time series data must be of type "INST-VAL" or "INST-CUM." A value in the resultant time series is set to missing if either the current or previous value in the original time series is missing (need to have two consecutive valid values). If the data type of the original data set is "INST-CUM" the resultant time series data set is assigned the type "PER-CUM," otherwise the data type does not change.

To compute the successive differences for time series data sets:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Successive Differences** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. Click **Compute**.

6.3.14 Time Derivative

The **Time Derivative** function computes the successive differences per unit time for a regular or irregular interval time series data set. For the time "t,"

$$TS2(t) = (TS1(t) – TS1(t-1)) / DT$$

where DT is the time difference between t and t-1. For the current form of the function, the units of DT are minutes.

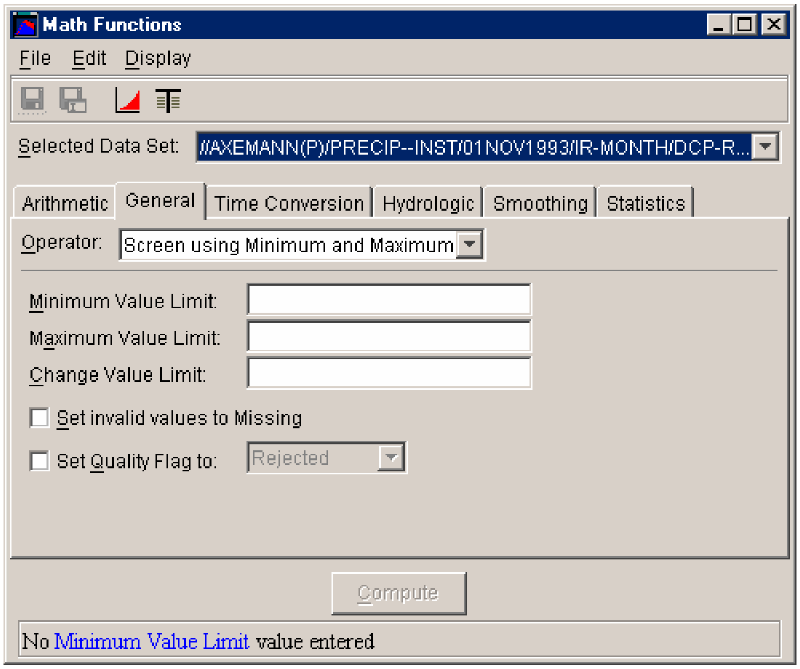
A value in the resultant time series is set to missing if either the current or previous value in the original time series is missing (need to have two consecutive valid values). By default, the data type of the resultant time series data set is assigned as "PER-AVER."

To compute the time derivative for time series data sets:

1. Choose the **Arithmetic** tab of the Math Functions Screen and select the **Time Derivative** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. Click **Compute**.

6.4 General Functions

The **General** tab (Figure 6.6) contains the functions Units Conversion, Round to Nearest Whole Number, Truncate to Whole Number, Round Off, Estimate Missing Values, Screen Using Maximum and Minimum, Screen Using Forward Moving Average, Merge Time Series, Merge Paired Data, and Generate Data Pairs.



Selected Data Set: //AXEMANN(P)/PRECIP--INST/01NOV1993/IR-MONTH/DCP-R...

Operator: Screen using Minimum and Maximum

Minimum Value Limit:

Maximum Value Limit:

Change Value Limit:

☐ Set invalid values to Missing

☐ Set Quality Flag to: Rejected

No Minimum Value Limit value entered

Figure 6.6 Math Functions--General Tab

6.4.1 Units Conversion

The **Units Conversion** function converts SI (metric) unit data to English units, or English unit data to SI units. The Units Conversion function may be applied to either time series or paired data sets.

To convert units for time series or paired data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Time Units Conversion** operator.
2. Choose the type of conversion by clicking **Convert to SI** or **Convert to English**.
3. From the **Selected Data Set** list, select one or more data sets for units conversion.
4. From the **Available Data Sets To Convert** list you may select additional data sets for units conversion. This list is affected by the type of units

conversion selected. If Convert to SI is selected, only data sets that are currently in English units will appear in the list. Similarly, if Convert to English is selected only data sets currently in SI units will appear.

5. Click **Compute** to perform the units conversion of the selected data sets.

6.4.2 Round to Nearest Whole Number

The **Round to Nearest Whole Number** function rounds values in a time series or paired data set to the nearest whole number.

The function rounds up the decimal portion of a number if equal to or greater than .5 and rounds down decimal values less than .5. For example:

10.5 is rounded to 11.

10.499 is rounded to 10.

-10.499 is rounded to -10.

-10.500 is rounded to -10.

-10.501 is rounded to -11.

The x-values in paired data sets are unaffected by the function, only the y-value data are rounded. For time series data sets, missing values are kept missing.

To round values in time series or paired data sets to the nearest whole number:

1. Choose the **General** tab of the Math Functions screen and select the **Round to Nearest Whole Number** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. Click **Compute**.

6.4.3 Truncate to Whole Number

The **Truncate to Whole Number** function truncates values in a time series or paired data set to the nearest whole number. For example:

10.99	is truncated to 10.
10.499	is truncated to 10.
-10.001	is truncated to -10.
-10.999	is truncated to -10.

The x-values in paired data sets are unaffected by the function, only the y-value data are truncated. For time series data sets, missing values are kept missing.

To truncate values in time series or paired data sets to the nearest whole number:

1. Choose the **General** tab of the Math Functions screen and select the **Truncate to Whole Number** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. Click **Compute**.

6.4.4 Round Off

The **Round Off** function rounds values in a time series or paired data set to a specified number of significant digits and/or power of tens place. For the power of tens place, -1 specifies rounding to one-tenth (0.1), while +2 rounds to the hundreds (100). For example, 1234.123456 will round to:

1230.0	for number of significant digits = 3,	power of tens place = -1
1234.1	for number of significant digits = 6,	power of tens place = -1
1234	for number of significant digits = 6,	power of tens place = 0
1230	for number of significant digits = 6,	power of tens place = 1

The x-values in paired data sets are unaffected by the function, only the y-value data are rounded. For time series data sets, missing values are kept missing.

To round values in time series or paired data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Round Off** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Number Significant Digits** box, set digits precision.
4. In the **Power of 10s Place** box, set the magnitude of 10 to which to round.
5. Click **Compute**.

6.4.5 Estimate Missing Values

The **Estimate Missing Values** function linearly interpolates estimates for missing values in a regular or irregular interval time series data set. Linear interpolation will occur for those portions of the time series data set where the number of consecutive missing values exceeds a specified user limit.

If the time series data set has type "INST-CUM", a special check box appears to optionally enable the following rules intended for cumulative precipitation:

- If the values bracketing the missing period are increasing with time, only interpolate if the number of successive missing values does not exceed the value of the user specified limit.
- If the values bracketing the missing period are decreasing with time, do not estimate any missing values.
- If the values bracketing the missing period are equal, then estimate any number of missing values

To estimate for missing values in time series data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Estimate Missing Values** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Maximum Consecutive Number of Missing** box, enter a value to set the limit for consecutive missing values allowed for interpolation.
4. If the time series data set is of type cumulative precipitation, select **Interpolate Cumulative Precip** to apply the modified rules for the interpolation of missing precipitation data.
5. Click **Compute** to perform the linear interpolation fill of the missing values in the data set.

6.4.6 Replace Specific Values

The **Replace Specific Values** function replaces all occurrences of a specified value with another. For example, you can use this function to change all occurrences of "1000" to "2000". If the data set contains precision information, then values within that precision limit will be replaced. For example, if the precision of the data set is "1" (number of digits to the right of the decimal), then a value of "12.3" will replace all values between "12.25" and "12.35". If no precision is set, then the values have to be exact to be replaced (not necessarily what is shown in a tabulation). This function works on both time-series and paired data sets.

To replace specific values in time series or paired data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Replace Specific Values** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Value to be replaced** box, enter the value in the data sets that you want to be replaced. If you want to replace missing values, leave this box empty.
4. In the **Value to replace with**, enter the new number that you want to replace the specified number with. If you want the value to be replaced to be set to missing, leave this box empty.
5. Click **Compute** to perform the operation.

6.4.7 Screen Using Minimum and Maximum

The **Screen Using Minimum and Maximum** function screens regular or irregular interval time series data sets for possible erroneous values based on user specified minimum-maximum value limits, and maximum absolute change. The maximum absolute change is tested only when the previous time series value is screened as valid. Only one test need to be specified.

Data values failing the screening test can be assigned a user specified quality flag and/or set to a specific value or to missing. The data sets may or may not contain prior quality flags. If the user specifies setting quality flags for screened data, they will be added to the resultant data sets if none already exists.

1. Choose the **General** tab of the Math Functions screen and select the **Screen Using Minimum and Maximum** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the

lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.

3. In the **Minimum Value Limit** and/or **Maximum Value Limit** boxes, enter the minimum and/or maximum valid value limits respectively.
4. In the **Change Value Limit** box, enter a value to set maximum absolute change allowed from the previous time series value if you want to test for this.
5. Select the **Set Invalid values to:** checkbox and fill enter the value you want to replace with for time series values failing the screening test. To set the value to missing, leave that box empty.
6. Select the **Set Quality Flag** box if the data quality flag is to be set for data values failing the screening test. If this box is checked, invalid data will be flagged with the quality selected in the list to the right. The available quality settings are: R(ejected), M(issing) and Q(uestionable).
7. Click **Compute**.

6.4.8 Screen with Forward Moving Average

The **Screen with Forward Moving Average** function screens a time series data set for possible erroneous values based on the change from the forward moving average computed at the previous point.

Data values failing the screening test are assigned a user specified quality flag and/or are set to the missing value. The data set may or may not currently have quality flags assigned. The forward moving average is computed over a user specified number of values. Missing values and values failing the screening test are not counted in the moving average and the divisor of the average is less one for each such value.

To screen data in time series data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Screen with Forward Moving Average** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the Number to Average Over box, enter a value to set the size of the moving average interval.
4. In the Change Value Limit box, enter a value to set maximum absolute change allowed from the forward moving average.
5. Select the Set Invalid Values to Missing box if time series values failing the screening test are to be set to the missing value.
6. Select the Set Quality Flag box if the data quality flag is to be set for data values failing the screening test. If this box is checked, invalid data will be flagged with the quality selected in the list to the right. The available quality settings are: R(ejected), M(issing) and Q(uestionable).

7. Click Compute to apply the function to the selected data set.

6.4.9 Merge Time Series

The **Merge Time Series** function merges data from one time series data set with another time series data set. The resultant time series data set includes all the data points in the two time series, except where the data points occur at the same time. When data points from the two data sets are coincident in time, valid values in the primary time series take precedence over valid values in the second selected time series. However if a coincident point is missing in the primary time series, a valid value in the second time series will be used for the data point in the resultant data set. If the value is missing for both time series data sets, the value is missing in the resultant data set.

The data sets for merging may be either regular or irregular time interval. The resultant data set is tested to determine if the times have a regular time interval. If not, it is typed as an irregular time interval data set.

Data of any type (i.e., "INST-CUM") or units may be merged with data of any other type or units. The resultant time series receives the data type and units of the primary time series.

To merge two time series data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Merge Time Series** operator.
2. From the **Selected Data Set** List, select the *primary* time series data set.
3. From the **Time Series** list, select the *second* time series data set for merging.
4. Click **Compute** to merge the two data sets.

6.4.10 Merge Paired Data

The **Merge Paired Data** function merges two paired data sets. The resultant paired data set includes all the paired data curves from the first data set and a single selected paired data curve or all curves from the second data set. The *x-values* for the two paired data sets *must match exactly*.

To merge two paired data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Merge Paired Data** operator.
2. From the **Selected Data Set** list, select the *primary* paired data set for merging.
3. From the **Paired Data** list, select the *second* paired data set for merging.
4. From the **Select Paired Data Curve** list, set the curve to be merged from the second paired data set (see section 6.2.1 "Selecting Paired Data Curves for Function Operations" for more details).
5. Click **Compute** to merge the two data sets.

6.4.11 Generate Data Pairs

The **Generate Data Pairs** function generates a paired data set by pairing values (by time) from two time series data sets. The data pairs in the paired data set may optionally be sorted by ascending x-value. An example use of the function would be to mate a time series record of stage to one of flow to generate a stage-flow paired data set.

The *times* in the two time series data sets *must match exactly*. If a value for a time is missing in either time series, no data value pair is formed and added to the paired data set.

Units and parameter type from the primary time series data set are assigned to the paired data set x-units and x-parameter type. Units and parameter type from the second time series are assigned to the paired data set y-units and y-parameter type.

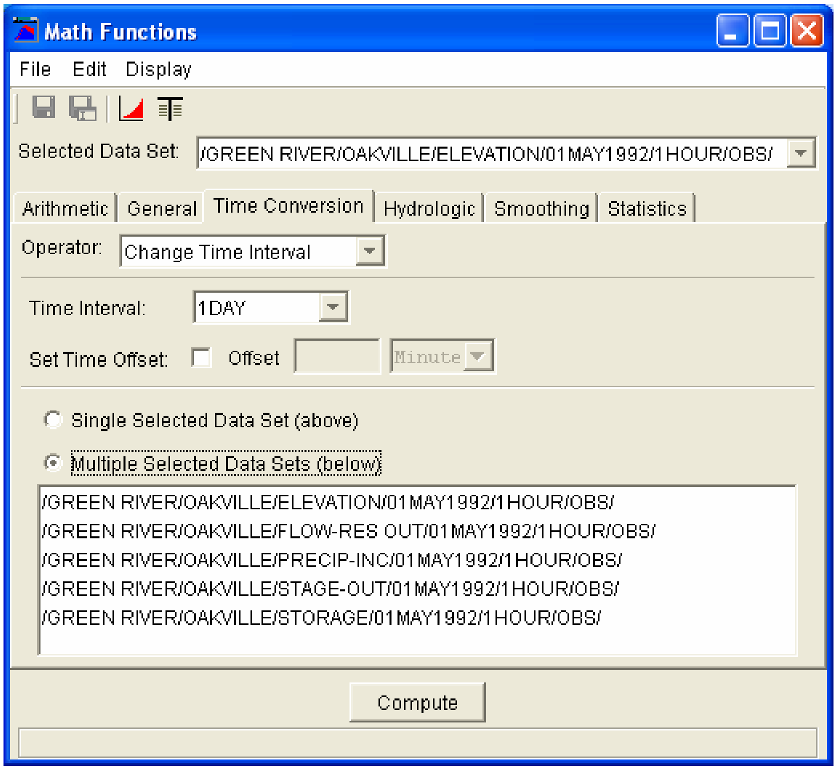
To generate a paired data set by pairing values in two time series data sets:

1. Choose the **General** tab of the Math Functions screen and select the **Generate Data Pairs** operator.
2. From the **Selected Data Set** list, select the primary time series data set. Time series values from this data set will comprise the x-values of the resultant paired data set.
3. From the **Data Set** list, select the second time series data set. Time series values from this data set will comprise the y-values of the resultant paired data set.
4. Check the **Sorted** box if the data pairs are to be sorted by ascending x-values.
5. Click **Compute**.

If time points in the two time series data sets do not match exactly, an error message will be posted and the function operation will not performed.

6.5 Time Conversion Functions

The **Time Conversion** tab (Figure 6.7) contains the functions Shift in Time, Change Time Interval, Irregular to Regular, Min/Max/Avg… Over Period, Snap Irregular to Regular, Regular to Irregular, To Irregular using Pattern, and Extract Time Series.



Selected Data Set: /GREEN RIVER/OAKVILLE/ELEVATION/01MAY1992/1HOUR/OBS/

Operator: Change Time Interval

Time Interval: 1DAY

Set Time Offset: □ Offset Minute

○ Single Selected Data Set (above)

● Multiple Selected Data Sets (below)

/GREEN RIVER/OAKVILLE/ELEVATION/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/FLOW-RES OUT/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/PRECIP-INC/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/STAGE-OUT/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/STORAGE/01MAY1992/1HOUR/OBS/

Figure 6.7 Math Functions--Time Conversion Tab

6.5.1 Min/Max/Avg/… Over Period

The **Min/Max/Avg/… Over Period** function generates a time series data set using a variety of functions from an existing irregular or regular interval time series data set. The functions are:

Interpolate at end of interval
Maximum over interval
Minimum over interval
Average over interval
Accumulation over interval
Integration over interval
Number of valid data over interval

where "interval" is a user selected time interval to evaluate the data over. This function is often used to compute information such as the maximum or minimum annual flow, total annual precipitation, and maximum and minimum daily temperatures. (See remarks following regarding computing values for water years.)

The resulting data can be put in an irregular-interval data set. By doing so, the date and time of each occurrence will be saved with the data. For example, if maximum annual flows are computed, the date and time of that maximum will be provided with an irregular-interval block. You should choose an irregular-interval block size, so that between 50 and 1,000 values are stored in a block. Daily data results should use a block size of "IR-YEAR". Annual results should use a block size of "IR-DECADE".

Besides choosing a standard interval, you can also select a **Custom Period Interval**. This provides you the capability to compute information over a period that is different from a standard storage interval, such as two-day averages, and maximum or accumulation over 2, 5 or 10 years. To compute using a custom interval, select **Custom…** from the **New Period Interval** pull-down box and then enter an integer interval value and select the **minute/hour/day/year** time increment for that value from the list further to the right. If you choose a custom period interval, the results must be stored in irregular-interval data sets.

The data type of the original time series data governs how values are interpolated. Data type "INST-VAL" (or "INST-CUM") considers the value to change linearly over the interval from the previous data value to the current data value. Data type "PER-AVER" considers the value to be constant at the current data value over the interval. Data type "PER-CUM" considers the value to increase from 0.0 (at the start of the interval) up to the current value over the interval. Interpolation of the three data types is illustrated in Figure 6.8.

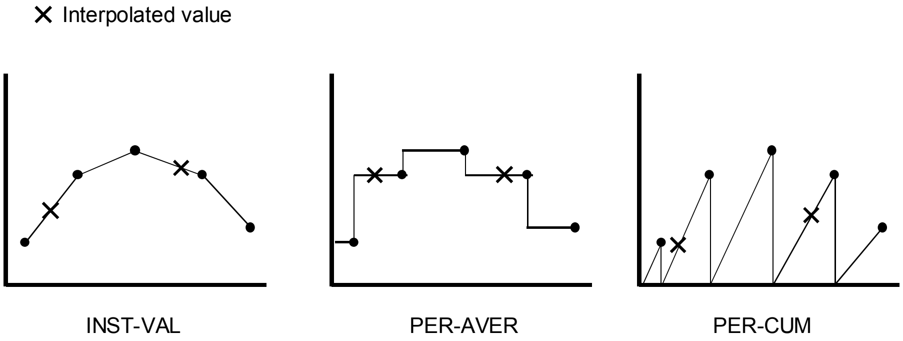


Figure 6.8 Interpolation of "INST-VAL," "PER-AVER" and "PER-CUM" Data

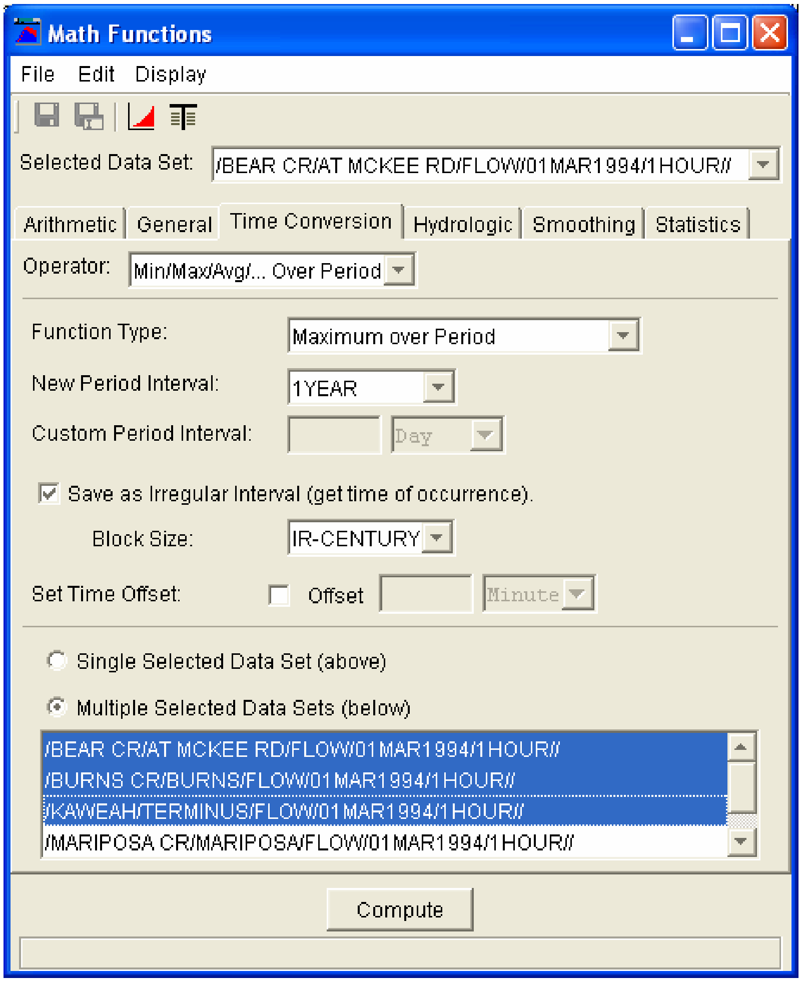
How interpolation is performed for a specific data type influences the computation of new time series values for the selected function. For example, if the data type is "INST-VAL," the function "Maximum over interval" is evaluated by:

- Finding the maximum value of the data points from the original time series that are inclusive in the new time interval.
- Linearly interpolate values at beginning and ending of the new time interval, and determine if these values represent the maximum over the interval.

Referring to the plots in Figure 6.8, the "Average over interval" function is applied to a time series by integrating the area under the curve between interpolated points and dividing the result by the interval time.

Times in the new time series may be shifted (offset) from the regular interval time by a user specified offset. As an example, the offset could be used to shift times in regular hourly interval data from the top of the hour to 6 minutes past the hour.

Figure 6.9 shows the appearance of the Math Functions screen for the **Min/Max/Avg/… Over Period** function.



Selected Data Set: /BEAR CR/AT MCKEE RD/FLOW/01MAR1994/1HOUR//

Operator: Min/Max/Avg/... Over Period

Function Type: Maximum over Period

New Period Interval: 1YEAR

Custom Period Interval: Day

☑ Save as Irregular Interval (get time of occurrence).

Block Size: IR-CENTURY

Set Time Offset: □ Offset Minute

○ Single Selected Data Set (above)

● Multiple Selected Data Sets (below)

/BEAR CR/AT MCKEE RD/FLOW/01MAR1994/1HOUR//
/BURNS CR/BURNS/FLOW/01MAR1994/1HOUR//
/KAWEAH/TERMINUS/FLOW/01MAR1994/1HOUR//
/MARIPOSA CR/MARIPOSA/FLOW/01MAR1994/1HOUR//

Figure 6.9 Screen for Min/Max/Avg/… Over Period Function

To compute the minimum, maximum, average, etc. of time series data sets:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Min/Max/Avg/… Over Period** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. From the **Function Type** list, select the computation to apply to the data.
4. From the **New Period Interval** list, select the interval period, or **Custom**, to compute values for.
5. If you select a **Custom** period, enter the integer period interval and select the **minute/hour/day/year** time increment for that value from the list further to the right.
6. Optionally, you may put the data in an irregular interval time series data set, which will retain the dates and times of each occurrence. You should choose an irregular-interval **block size**, so that between 50 and 1,000

values are stored in a block. Daily data results should use a block size of "IR-YEAR". Annual results should use a block size of "IR-DECADE".

7. Optionally, you may specify a **Time Offset** to offset the new points in time from the standard interval time. For example, if you select a daily interval for the Time Interval, you may use the option to offset time points to be at 6 a.m. instead of at midnight. If the **Specify Offset** box is checked, enter an integer number in the box to the right and select the **minute/hour/day** time increment from the list further to the right. If the Specify Offset box is unchecked, times in the resultant data sets will fall on the regular interval time (typical setting).
8. Click **Compute**.

6.5.2 Copy in Time

The **Copy in Time** function copies the data set and sets a new start time with a specified increment from the original or to start at a specified date and time. Both the original and the copied data values are not changed, except where the two data sets overlap. The difference between the **Copy in Time** function and the **Shift in Time** function is that the shift function deletes the original data before storing the shifted data, whereas the copy function does not. Data sets may be regular or irregular interval time series data.

To copy time series data sets to another time:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Copy in Time** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Copy by Amount** box, enter the amount of the time to increment the data sets by (integer values only) and then select **Minute**, **Hour**, **Day** or **Year**. To increment data backwards in time, use a negative amount for the time shift.
4. Alternatively, you may select the **Copy to Date/Time** radio button and specify the date and time to copy the first value in each data set to.

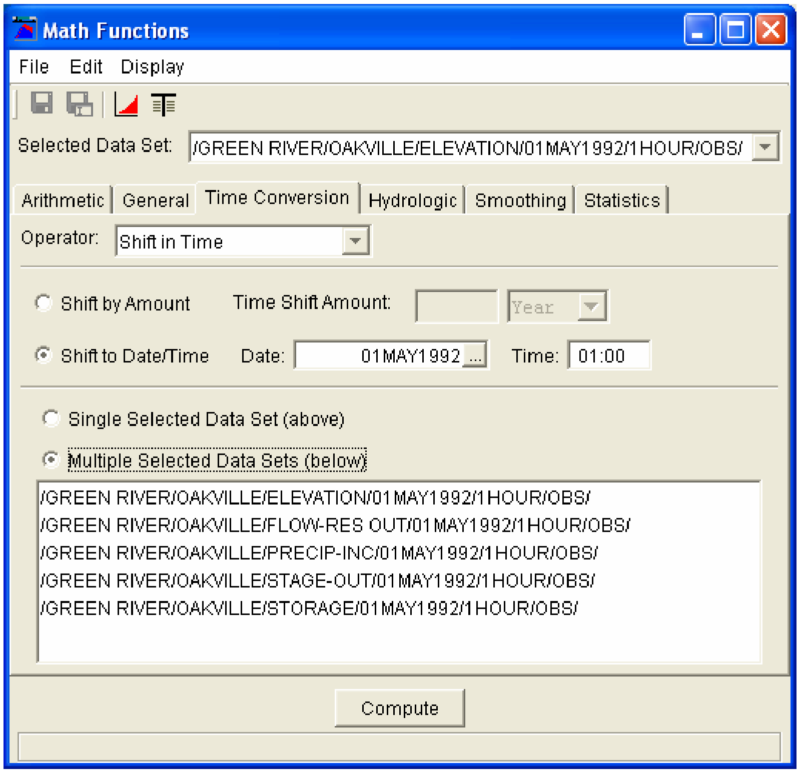
6.5.3 Shift in Time

The **Shift in Time** function shifts the times in time series data sets by a specified increment or to start at a specified date and time. The original data values are removed from the file when the shifted data is stored. To accomplish this, the data sets must be saved immediately following the operation. For example, if you shift data by one year from 1965 to 1966, data will exist for 1966 and not 1965. To retain the original data, use **the Copy in**

Time function. Data sets may be regular or irregular interval time series data.

To shift times in time series data sets:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Shift in Time** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Time Shift Amount** box, enter the amount of the time to shift the data sets by (integer values only) and then select **Minute**, **Hour**, **Day** or **Year**. To shift data backwards in time, use a negative amount for the time shift.
4. Alternatively, you may select the **Shift to Date/Time** radio button and specify the date and time to shift the first value in each data set to.



Selected Data Set: /GREEN RIVER/OAKVILLE/ELEVATION/01MAY1992/1HOUR/OBS/

Operator: Shift in Time

○ Shift by Amount Time Shift Amount: Year

● Shift to Date/Time Date: 01MAY1992 Time: 01:00

○ Single Selected Data Set (above)

● Multiple Selected Data Sets (below)

/GREEN RIVER/OAKVILLE/ELEVATION/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/FLOW-RES OUT/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/PRECIP-INC/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/STAGE-OUT/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/STORAGE/01MAY1992/1HOUR/OBS/

Figure 6.101 Screen for Shift in Time Function

6.5.4 Change Time Interval

The **Change Time Interval** function interpolates or extracts values from existing regular interval time series data sets to new regular interval time series data sets with a different time interval.

Whether the time series data type is "INST-VAL," "INST-CUM," "PER-AVER," or "PER-CUM" controls how interpolation is performed. Interpolated values are derived from "INST-VAL" or "INST-CUM" data using linear interpolation. Values are derived from "PER-AVER" data by computing the period average value over the new time interval. Values are derived from "PER-CUM" data by computing the period cumulative value over the new time interval.

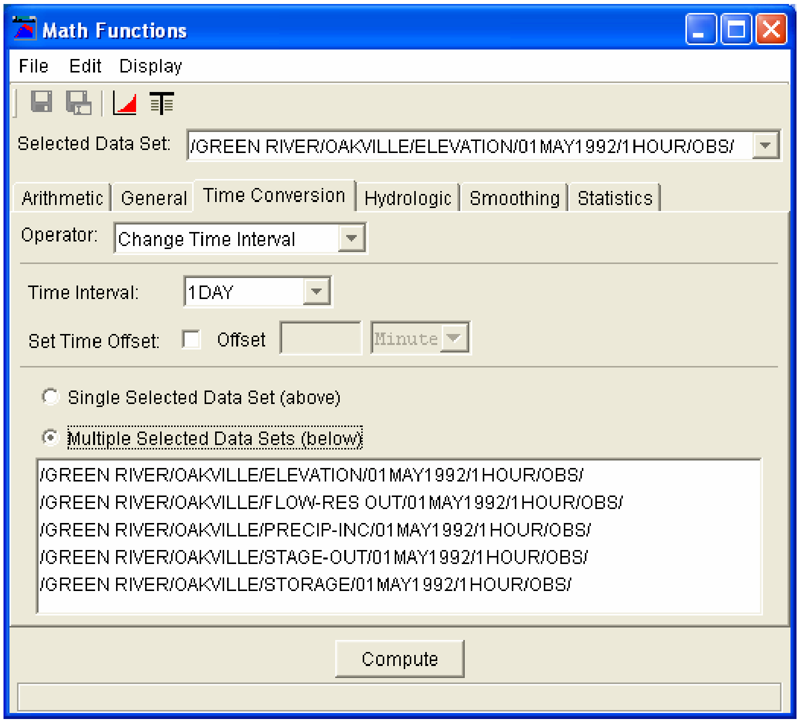
For example, if an original data set is hourly data and the new regular interval data set is to have a six-hour time interval:

1. The value for "INST-VAL" or "INST-CUM" type data is computed from the linear interpolation of hourly points bracketing the new six-hour time point.
2. The value for "PER-AVER" type data is computed from the period average value over the six-hour interval.
3. The value for "PER-CUM" type data is computed from the accumulated value over the six-hour interval.

The treatment of missing value data is also dependent upon data type. Interpolated "INST-VAL" or "INST-CUM" points must be bracketed or coincident with valid (not missing) values in the original time series, otherwise the interpolated values are set as missing. Interpolated "PER-AVER" or "PER-CUM" data must contain all valid values over the interpolation interval; otherwise the interpolated value is set as missing.

To change the time interval in time series data sets:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Change Time Interval** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. From the **Time Interval** list, select the new time interval for the data sets.
4. Optionally, you may specify a **Time Offset** to offset the new points in time from the standard interval time. For example, if you select a daily interval for the Time Interval, you may use the option to offset time points to be at 6 a.m. instead of at midnight. If the **Specify Offset** box is checked, enter an integer number in the box to the right and select the **minute/hour/day** time increment from the list further to the right. If the Specify Offset box is unchecked, times in the resultant data sets will fall on the regular interval time (typical setting).
5. Click the **Compute** button to perform the interpolation.



Selected Data Set: /GREEN RIVER/OAKVILLE/ELEVATION/01MAY1992/1HOUR/OBS/

Operator: Change Time Interval

Time Interval: 1DAY

Set Time Offset: □ Offset

○ Single Selected Data Set (above)

● Multiple Selected Data Sets (below)

/GREEN RIVER/OAKVILLE/ELEVATION/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/FLOW-RES OUT/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/PRECIP-INC/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/STAGE-OUT/01MAY1992/1HOUR/OBS/
/GREEN RIVER/OAKVILLE/STORAGE/01MAY1992/1HOUR/OBS/

Figure 6.102 Screen for Change Time Interval Function

6.5.5 Irregular to Regular

The **Irregular to Regular** function interpolates or extracts values from existing irregular interval time series data sets to create new regular interval time series data sets.

Whether the time series data type is "INST-VAL," "INST-CUM," "PER-AVER," or "PER-CUM" controls how the interpolation is performed. Interpolated values are derived from "INST-VAL" or "INST-CUM" data using linear interpolation. Values are derived from "PER-AVER" data by computing the period average value over the new regular time interval. Values are derived from "PER-CUM" data by computing the period cumulative value over the new regular time interval.

For example, if the original data set has data that typically occurs several times in a six-hour period, and the new regular interval data set is to have a six-hour time interval:

- The value for "INST-VAL" or "INST-CUM" type data is computed from the linear interpolation of points bracketing the new six-hour time point.
- The value for "PER-AVER" type data is computed from the period average value over the six-hour interval.

- The value for "PER-CUM" type data is computed from the accumulated value over the six-hour interval.

To convert irregular interval time series data to regular interval data:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Irregular to Regular** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. From the **Time Interval** list, select the time interval for the data sets.
4. Optionally, you may specify a **Time Offset** to offset the new points in time from the standard interval time. For example, if you select a daily interval for the Time Interval, you may use the option to offset time points to be at 6 a.m. instead of at midnight. If the **Specify Offset** box is checked, enter an integer number in the box to the right and select the **minute/hour/day** time increment from the list further to the right. If the Specify Offset box is unchecked, times in the resultant data sets will fall on the regular interval time (typical setting).
5. Click the **Compute** button to perform the interpolation.

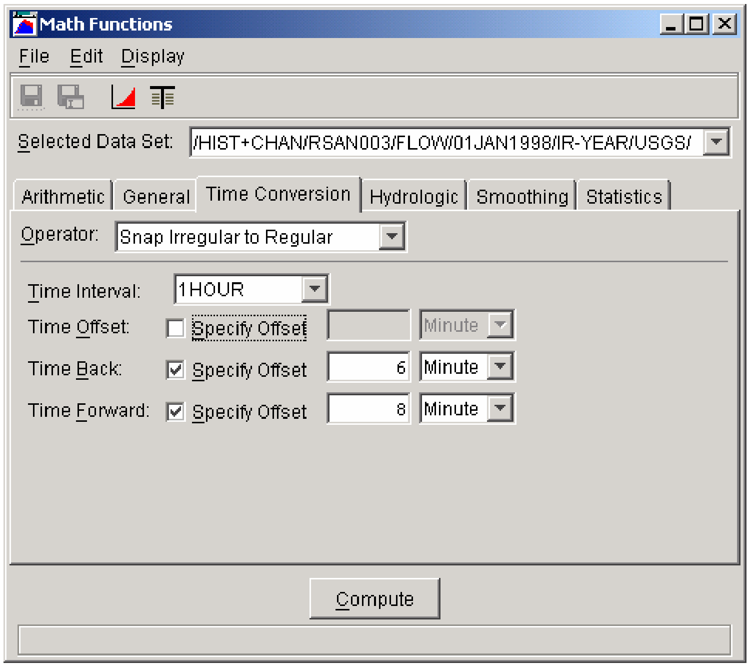
6.5.6 Snap Irregular to Regular

The **Snap Irregular to Regular** function derives new regular interval time series data sets from existing irregular or regular interval time series data sets. Data from original time series will be "snapped" to a user specified regular interval time if the time of the original data falls within the time window tolerance set by the user. A snap just changes the time of the data values and does no interpolation. For example, a time series record from a gauge recorder collects readings 6 minutes past the hour. The Snap Irregular to Regular function is often used to "snap" or shift the time points to the top of the hour.

Times in resultant time series may be shifted (offset) from the regular interval time by a user specified offset. As an example, the offset could be used to shift times in regular hourly interval data from the top of the hour to 6 minutes past the hour.

By default values in the resultant regular interval time series data sets are set to missing unless matched to times in the original time series data set (within the time window tolerance).

Figure 6.103 shows the appearance of the Math Functions screen for the Snap Irregular to Regular function.



Selected Data Set: /HIST+CHAN/RSAN003/FLOW/01JAN1998/IR-YEAR/USGS/

Operator: Snap Irregular to Regular

Time Interval: 1HOUR

Time Offset: □ Specify Offset Minute

Time Back: ☑ Specify Offset 6 Minute

Time Forward: ☑ Specify Offset 8 Minute

Figure 6.103 Screen for Snap Irregular to Regular Function.

To snap irregular time interval data to regular times:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Snap Irregular to Regular** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. From the **Time Interval** list, select the regular interval time period for the resultant time series data sets.
4. Optionally, you may specify a **Time Offset** to offset the new points in time from the standard interval time. For example, if you select a daily interval for the Time Interval, you may use the option to offset time points to be at 6 a.m. instead of at midnight. If the **Specify Offset** box is checked, enter an integer number in the box to the right and select the **minute/hour/day** time increment from the list further to the right. If the Specify Offset box is unchecked, times in the resultant data sets will fall on the regular interval time (typical setting).
5. Use the **Time Back** and **Time Forward** offsets to define a time window around the regular interval time. If a time point from the selected time series data set falls within the time window, the data value is applied to the new regular interval time point. These offsets are set in the same manner as the **Time Offset** above.
6. Click **Compute**.

6.5.7 Regular to Irregular

The **Regular to Irregular** function derives new irregular interval time series data sets from existing regular interval time series data sets. No conversion, time shifting, or other operations are applied to the data. Embedded missing data values are retained in the irregular interval data sets.

The **Irregular Block Size** dictates how the data is stored in the HEC-DSS file. You should choose an irregular-interval **block size**, so that between 50 and 1,000 values are stored in a block. Data that occurs approximately on the average of once an hour should use a block size of "IR-MONTH". Data that occurs approximately on the average of once a day should use a block size of "IR-YEAR". Annual values should use a block size of "IR-DECADE".

To convert regular interval time series data to irregular interval data:

1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Regular to Irregular** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. Select the **Irregular Block Size** for the data sets. Choose an irregular-interval **block size**, so that between 50 and 1,000 values are stored in a block. Data that occurs approximately on the average of once an hour should use a block size of "IR-MONTH". Data that occurs approximately on the average of once a day should use a block size of "IR-YEAR". Annual values should use a block size of "IR-DECADE".
4. Click the **Compute** button.

6.5.8 To Irregular using Pattern

The **To Irregular using Pattern** function generates a new time series data set from an existing irregular or regular interval time series data set. The times for the new time series are defined by the times of a second time series data set. Values for the new time series are computed from the original time series data using one of seven available functions. The functions are:

Interpolate at end of period
Maximum over period
Minimum over period
Average over period
Accumulation over period
Integration over period
Number of valid data over period

where "period" is the time between data points in the pattern data set.

The data type of the original time series data governs how values are interpolated. Data type "INST-VAL" (or "INST-CUM") considers the value to change linearly over the interval from the previous data value to the current data value. Data type "PER-AVER" considers the value to be constant at the current data value over the interval. Data type "PER-CUM" considers the value to increase from 0.0 (at the start of the interval) up to the current value over the interval. Interpolation of the three data types is illustrated in Figure 6.104.

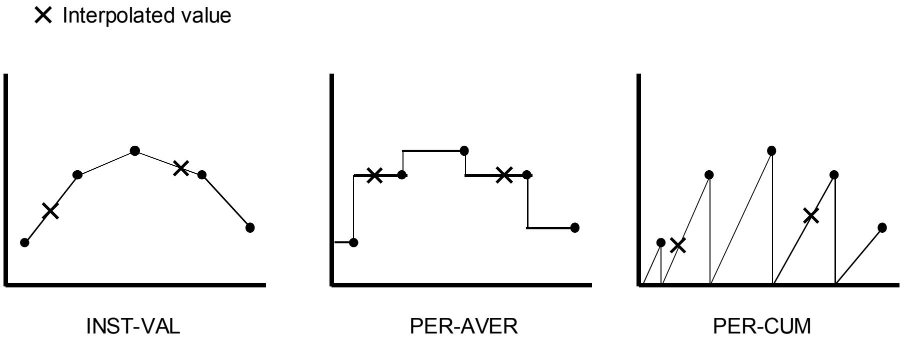
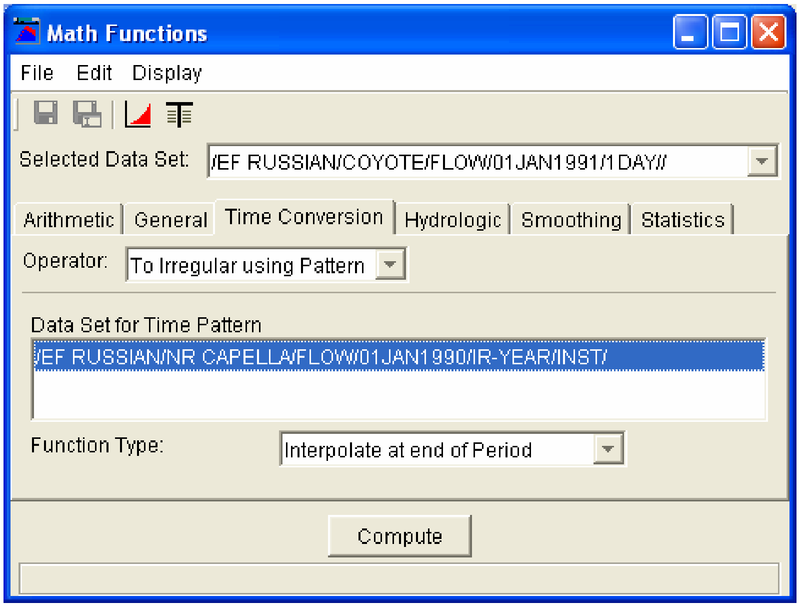


Figure 6.104 Interpolation of "INST-VAL," "PER-AVER" and "PER-CUM" Data

How interpolation is performed for a specific data type influences the computation of new time series values for the selected function. For example, if the data type is "INST-VAL," the function "Maximum over interval" is evaluated by:

- Finding the maximum value of the data points from the original time series that are inclusive in the new time interval.
- Linearly interpolate values at beginning and ending of the new time interval, and determine if these values represent the maximum over the interval.

Referring to the plots in Figure 6.104, the "Average over interval" function is applied to a time series by integrating the area under the curve between interpolated points and dividing the result by the interval time.


Math Functions
File Edit Display
Arithmetic General Time Conversion Hydrologic Smoothing Statistics
Compute

Selected Data Set: /EF RUSSIAN/COYOTE/FLOW/01JAN1991/1DAY//

Operator: To Irregular using Pattern

Data Set for Time Pattern
/EF RUSSIAN/NR CAPELLA/FLOW/01JAN1990/IR-YEAR/INST/

Function Type: Interpolate at end of Period

Figure 6.105 Screen for Transforming To Irregular using Pattern Function

Figure 6.105 shows the appearance of the Math Functions screen for the Transform to Irregular interval function.

To transform to an irregular interval time series data set:

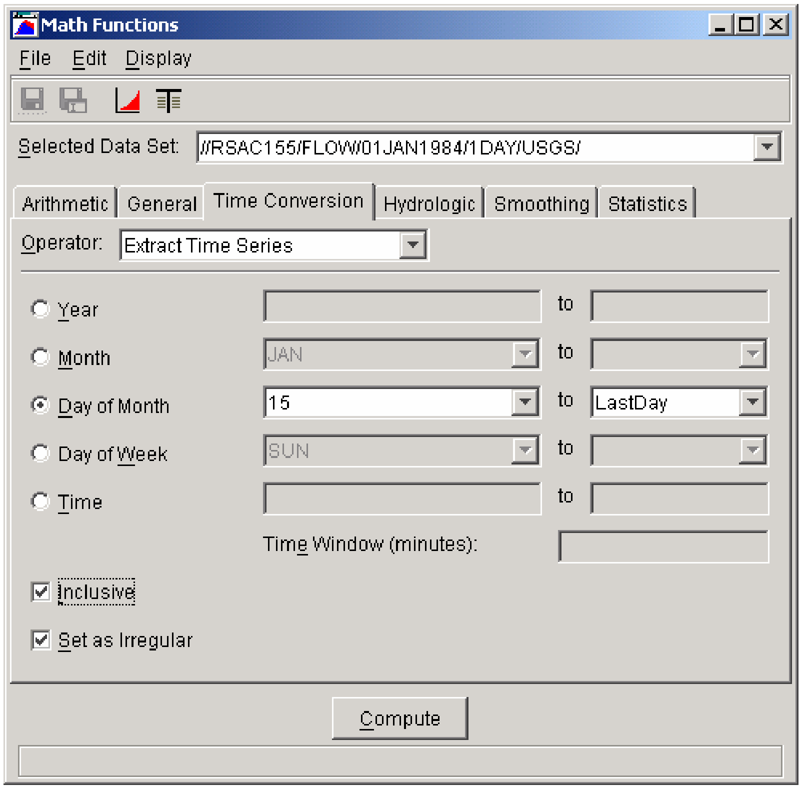
1. Choose the **Time Conversion** tab of the Math Functions screen and select the **To Irregular using Pattern** operator.
2. From the **Selected Data Set** list, select a time series data set to apply the function.
3. From the **Data Set for Time Pattern** list, select a time series data set to provide the time pattern for the resultant time series.
4. From the **Function Type** list, select the method for computing new time series values.
5. Click **Compute**.

6.5.9 Extract Time Series

The **Extract Time Series** function selects/extracts data points from a regular or irregular interval time series data set based on user defined time specifications. For example, the function may be used to extract values observed every day at noon from hourly interval data.

Data may be extracted on the basis of one of the following time specifications: year, month of the year (January, February, etc.), day of month, day of week (Sunday, Monday, etc.), or time of day.

Figure 6.106 shows the appearance of the Math Functions screen for the Extract Time Series function.



Selected Data Set: //RSAC155/FLOW/01JAN1984/1DAY/USGS/

Operator: Extract Time Series

○ Year to

○ Month JAN to

● Day of Month 15 to LastDay

○ Day of Week SUN to

○ Time to

Time Window (minutes):

☑ Inclusive

☑ Set as Irregular

Figure 6.106 Screen for Extract Time Series Function

To derive a new time series data set from the selected times of another time series data set:

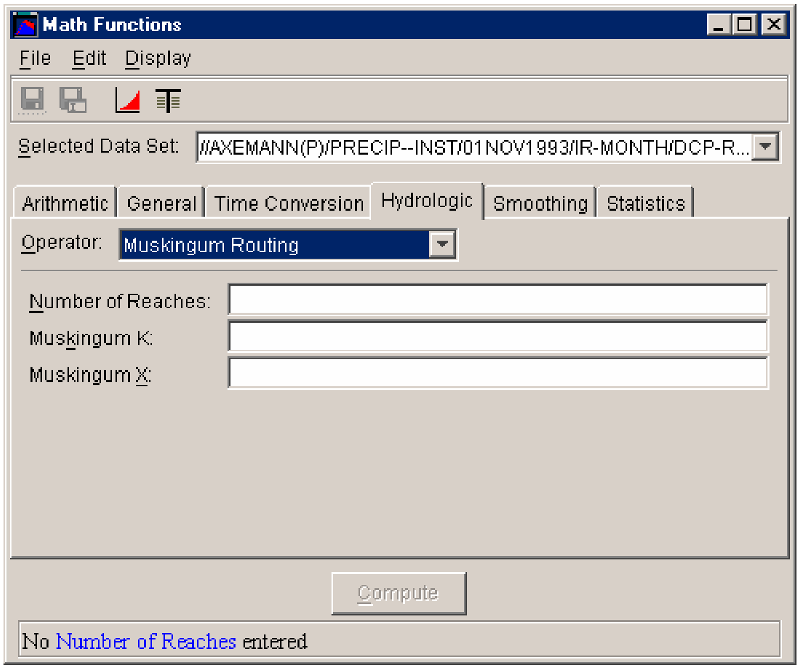
1. Choose the **Time Conversion** tab of the Math Functions screen and select the **Extract Time Series** operator.
2. From the **Selected Data Se**t list, select a time series data set to apply the function.
3. Choose one of the time levels, **Year**, **Month**, **Day of Month**, **Day of Week**, or **Time** of day, and complete the boxes to the right. Entering or selecting a value in only the left-most box sets a time specification for data extraction to an individual year, month, day, or time of day. Fill in both boxes to specify a time range for data extraction.
4. Click **Year** to extract data for an individual year or range of years. To extract data for an individual year, fill the left-most box in with the desired year. To extract data for a range of years fill the left box with the beginning year and the right box with the ending year.
5. Click **Month** to extract data for an individual month of the year or range of months. Use the lists to set the month boxes. The list contains the months of the year, JAN to DEC. Set only the left-most box to extract data for a single month of the year. Set both boxes to extract data for a range of months. The range of months can be set to encompass the end of the year, that is "OCT to FEB" is a valid setting.
6. Click **Day of Month** to extract data for an individual day or range of day of month days. The days are set using the two lists to the right. As above, the lists can be used to specify a single day of month, using the left-most pull down, or a range of days, using both lists. The day of month values range from 1 to 31, and "LastDay," the last day of the month, which may vary by month. The range of days can be set to encompass the end of the month. For example, for "27 to 4," data will be extracted from the 27th of one month to the 4th of the next month.
7. Click **Day of Week** to extract data for an individual or range of day of the weekdays. Select day from the two lists to the right. As above, you may use the lists to specify one day, using the left-most list, or a range of days, using both lists. The day of week values range from SUN to SAT. The range of days can be set to encompass the end of the week; that is, "SAT to MON" is a valid setting.
8. Click **Time** to extract data by the time of day. An individual time is specified by completing the left-most box only. Complete both boxes to specify a time of day range for data extraction. The time specification employs the standard four-digit military style 24-hour format (e.g. "2300" or "0310 to 0600"). The time range can encompass the end of day, that is "2200 to 0330" is a valid setting. The optional **Time Window** box is applied to the time of day extraction only. The Time Window is an integer value in minutes, used to extend the beginning and ending of the time of day period for data extraction. For example, with a time of day extraction time of "0300" and a Time Window of "10 minutes," data will

be extracted from the selected time series if times falls within in the period 0250 to 0310.

9. By default, the data is extracted for the time points falling within the time interval set above. Click to clear the **Inclusive** check box at the bottom of the screen if you want to use the time specifications set above to exclude data for times within the specified range, and to include data falling outside the range. For example, if `Inclusive` is checked and the time range is set to "JAN-MAR," the extracted data will include all data in the months January through March for all the years of time series data. If `Inclusive` is unchecked, the extracted data will cover the period April through December (is exclusive of the period January through March).
10. Select the **Set as Irregular** checkbox (recommended) to ensure the resultant time series data set is identified as irregular interval time series data. If the box is not checked, the function will attempt to determine if the extracted data set can be classified as regular interval time series data.

6.6 Hydrologic Functions

The **Hydrologic** tab (Figure 6.107) contains the functions Muskingum Routing, Straddle Stagger Routing, Modified Puls Routing, Rating Table, Reverse Rating Table, Two Variable Rating Table, Decaying Basin Wetness, Shift Adjustment, Period Constants, Multiple Linear Regression, Conic Interpolation, Polynomial, Polynomial with Integral, and Flow Accumulator Gage Processor.



Selected Data Set: //AXEMANN(P)/PRECIP--INST/01NOV1993/IR-MONTH/DCP-R...

Operator: Muskingum Routing

Number of Reaches:

Muskingum K:

Muskingum X:

No Number of Reaches entered

Figure 6.107 Math Functions--Hydrologic Tab

6.6.1 Muskingum Routing

The **Muskingum Routing** function routes a regular interval time series data set by the Muskingum hydrologic routing method.

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Muskingum Routing** operator.
2. From the **Selected Data Set** list, select the time series data set for routing.
3. In the **Number of Reaches** box, specify the number of routing subreaches.
4. In the **Muskingum K** box, enter the Muskingum "K" routing parameter (travel time, in hours).
5. In the **Muskingum X** box, enter the Muskingum "x" routing parameter. The Muskingum "x" value cannot be less than 0.0 (maximum attenuation) or greater than 0.5 (no attenuation).
6. Click **Compute** to route the selected time series data set.

An error message will be displayed if a compute is attempted using an invalid Muskingum "x" value, and no routing will be performed.

The set of Muskingum routing parameters entered may potentially produce numerical instabilities in the routed time series. The parameters are first checked by the function for possible instabilities before the routing proceeds. If instabilities are possible, a warning message box will appear presenting the details of the instabilities. Click the **Yes** button in the message box to proceed with the compute, or click **No** to cancel the operation.

6.6.2 Straddle Stagger Routing

The **Straddle Stagger Routing** function routes a regular interval time series data set by the Straddle Stagger hydrologic routing method.

To route a time series data set by the Straddle Stagger hydrologic routing method:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Straddle Stagger Routing** operator.
2. From the **Selected Data Set** list, select the time series data set for routing.
3. In the **Number to Average** box, enter the number of ordinates to average over (Straddle).
4. In the **Number to Lag** box, enter the number of ordinates to lag (Stagger).
5. In the **Number of SubReaches** box, enter the number of routing subreaches.
6. Click **Compute** to route the selected time series data set.

6.6.3 Modified Puls Routing

The **Modified Puls Routing** function routes a regular interval time series data set by the Modified Puls hydrologic routing method.

With the Modified Puls method, outflow from a routing reach is a unique function of storage. The storage-discharge relationship is entered into the function as a paired data set, where the x-values are storage and the y-values are discharge.

To route a time series data set by the Modified Puls hydrologic routing method:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Modified Puls Routing** operator.
2. From the **Selected Data Set** list, select the time series data set for routing.
3. From the **Storage-Discharge Paired Data Set** list, select the paired data set containing the storage-discharge table. Only paired data sets will appear in this list, and only one data set at a time can be selected.
4. In the **Number of SubReaches** box, enter the number of routing subreaches.
5. In the **Muskingum X** box, enter the Muskingum "x" routing parameter. The Muskingum "x" value cannot be less than 0.0 or greater than 0.5.

Enter 0.0 to route by the Modified Puls method, or a value greater than 0.0 to apply the Working R&D method.

6. Click **Compute** to route the select time series data set.

6.6.4 Rating Table

The **Rating Table** function transforms/interpolates values in a time series data set using the rating table x-y values stored in a paired data set. For example, the Rating Table function can be used to transform stage values to flow values using a stage-flow rating table.

The units and parameter type of the resultant time series data set are duplicated from the y-units and y-parameter type of the rating table paired data set. For example, if a stage-flow rating table has a y-data parameter type of "FLOW" and y-units of "cfs", the resultant time series data set will have the data parameter type "FLOW" and the units "cfs."

To derive a new time series data set from the rating table interpolation based on the selected time series data set:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Rating Table** operator.
2. From the **Selected Data Set** list, select a time series data set for rating table interpolation. If the selected data set is not a time series, the remaining input boxes on the screen will be unavailable.
3. From the **Rating Table** list, select the paired data set containing the rating table information. Only paired data sets will appear in this list, and you may select only one data set at a time.
4. Click **Compute** to perform the rating table interpolation of the selected time series data set.

6.6.5 Reverse Rating Table

The **Reverse Rating Table** function transforms/interpolates values in a time series data set using the reverse of the rating table stored in a paired data set. For example, the Reverse Rating Table function can be used to transform flow values to stage values using a stage-flow rating table.

The units and parameter type of the resultant time series data set are duplicated from the x-units and x-parameter type of the rating table paired data set. For example, if a stage-flow rating table has a x-data parameter type of "STAGE" and x-units of "ft," the resultant time series data set will have the data parameter type "STAGE" and the units "ft."

To derive a new time series data set from the reverse rating table interpolation of the selected time series data set:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Reverse Rating Table** operator.
2. From the **Selected Data Set** list, select a time series data set for reverse rating table interpolation. If the selected data set is not a time series, the remaining input boxes on the screen will be unavailable.
3. From the **Rating Table** list, select the paired data set containing the rating table information. Only paired data sets will appear in this list, and you may select only one data set at a time.
4. Click **Compute** to perform the reverse rating table interpolation of the selected time series data set.

6.6.6 Two Variable Rating Table

The **Two Variable Rating Table** function computes new time series values from the input of two other independent time series data sets. The functional relationship is specified by a table of values contained in a paired data set having multiple sets of x-y curves.

As an example, reservoir release is a function of both the gate opening height and reservoir elevation (Figure 6.108).

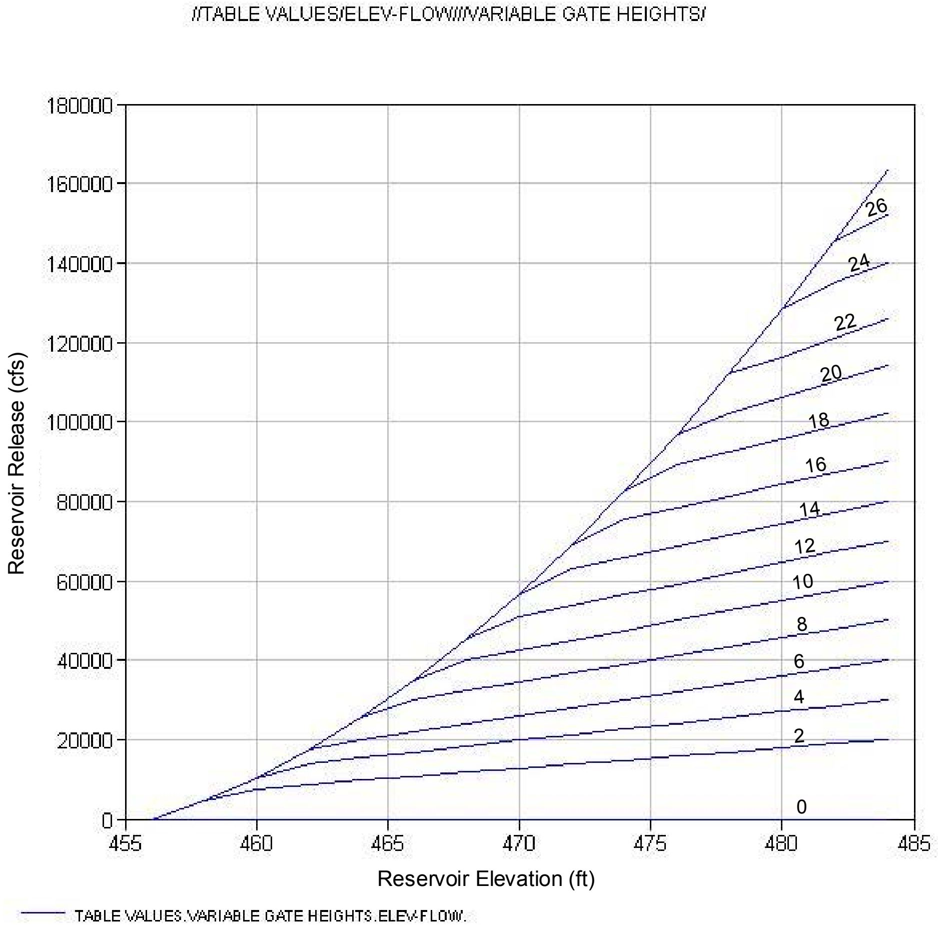


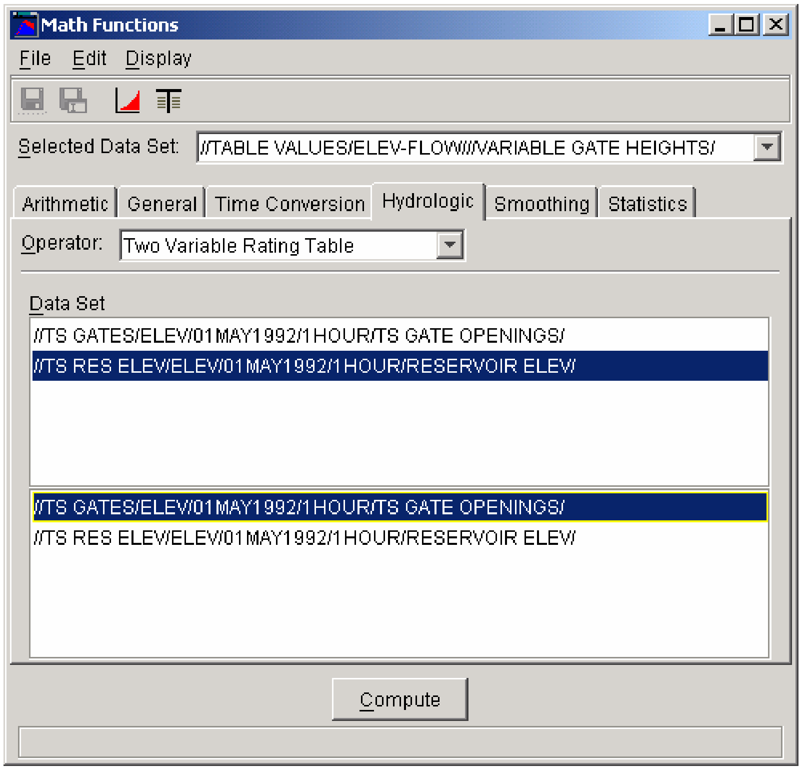
Figure 6.108 Example of two variable rating table paired data, reservoir release as a function of reservoir elevation and gate opening height (curve labels).

For each gate opening height, there is a reservoir elevation-reservoir release curve, where reservoir elevation is the independent variable (x-values) and reservoir release is the dependent variable (y-values) of a paired data set. Each paired data curve has a curve label. In this case, the curve label is assigned the gate opening height. Using the paired data set shown in Figure 6.108, you may employ the Two Variable Rating Table function to interpolate time series values of reservoir elevation and gate opening height to develop a time series of reservoir release.

Times for the two input time series data sets must match. Curve labels must be set for curves in the rating table paired data set and must be interpretable as numeric values.

By default, the units and parameter type of the resultant time series data set are duplicated from the y-units and y-parameter type of the rating table paired data set.

Figure 6.109 shows the appearance of the Math Functions screen for the Two Variable Rating Table function.



Selected Data Set: //TABLE VALUES/ELEV-FLOW///VARIABLE GATE HEIGHTS/

Operator: Two Variable Rating Table

Data Set

//TS GATES/ELEV/01MAY1992/1HOUR/TS GATE OPENINGS/
//TS RES ELEV/ELEV/01MAY1992/1HOUR/RESERVOIR ELEV/

//TS GATES/ELEV/01MAY1992/1HOUR/TS GATE OPENINGS/
//TS RES ELEV/ELEV/01MAY1992/1HOUR/RESERVOIR ELEV/

Figure 6.109 Screen for Two Variable Rating Table Function.

To derive a new time series data set from the two variable rating table interpolation of two other time series data sets:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Two Variable Rating Table** operator.
2. From the **Selected Data Set** list, select the paired data set containing the rating table. If the selected data set is not a paired data set, the remaining input boxes on the screen will be unavailable.
3. From the two lists shown below the **Data Set** title, select the two input time series data sets.
   - Use the top list to select the time series data set matching the **x-values parameter** of the rating table. In Figure 6.109, this is a time series data set of reservoir elevation.
   - Use the bottom list to select the time series data set matching the **curve labels parameter**. In Figure 6.109 this is a time series data set of gate opening height. You may select only one data set from a list.
4. Once the data sets are selected, click **Compute** to apply the Two Variable Rating Table function**.**

6.6.7 Decaying Basin Wetness

The **Decaying Basin Wetness** function computes a time series of decaying basin wetness parameters from a regular interval time series data set of incremental precipitation by:

TSPar(t) = Rate * TSPar(t-1) + TSPrecip(t)

where Rate is the decay rate and $0 < \text{Rate} < 1$.

The first value of the resultant time series, TSPar(1), is set to the first value in the precipitation time series, TSPrecip (1). Missing values in the precipitation time series are zero for the above computation.

To compute a time series of decaying basin wetness parameters:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Decaying Basin Wetness** operator.
2. From the **Selected Data Set** list, select the time series data set of incremental precipitation.
3. Enter the **Decay Rate**. This value must be greater than 0.0 and less than 1.0.
4. Click **Compute**.

6.6.8 Shift Adjustment

The **Shift Adjustment** function linearly interpolates values in the primary time series data set to the times defined by a second time series data set. If times in the new time series precede the first data point in the primary time series, the value for these times is set to 0.0. If times in the new time series occur after the last data point in the primary time series, the value for these times is set to the value of the last point in the primary time series. Interpolation of values with the Shift Adjustment function is shown in Figure 6.110

Both time series data sets may be regular or irregular interval. Interpolated points must be bracketed or coincident with valid (not missing) values in the original time series, otherwise the values are set as missing.

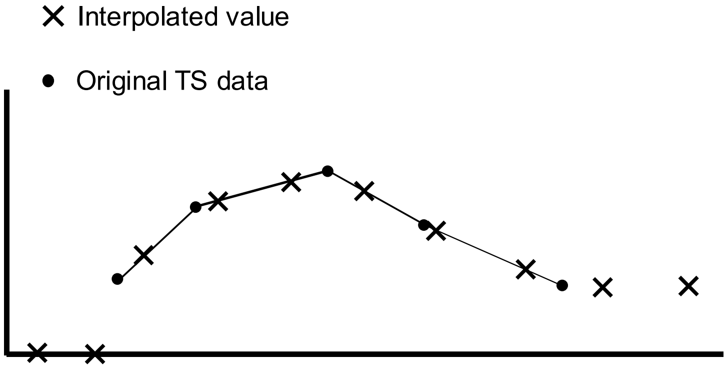


Figure 6.110 Interpolation of time series values using Shift Adjustment function.

To generate a new time series of shift adjustments:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Shift Adjustment** operator.
2. From the **Selected Data Set** list, select the *primary* time series data set. This data set has the *values* for the interpolation.
3. From the **Data Set for Time Pattern** list, select the *second* time series data set. This data set has the *times* for the interpolation.
4. Click **Compute**.

6.6.9 Period Constants

The **Period Constants** function applies *values* in the *primary* time series data set to the *times* defined by a *second* time series data set. Both time series data sets may be regular or irregular interval. Values in the new time series are set according to:

$$ts1(j) \leq tsnew(i) < ts1(j+1)\,, \qquad TSNEW(i) = TS1(j)$$

where ts1 is the time in the primary time series, TS1 is the value in the primary time series, tsnew is the time in the new time series, TSNEW is the value in the new time series.

If times in the new time series precede the first data point in the primary time series, the value for these times is set to missing. If times in the new time series occur after the last data point in the primary time series, the value for these times is set to the value of the last point in the primary time series. Interpolation of values with the Period Constants function is shown in Figure 6.111

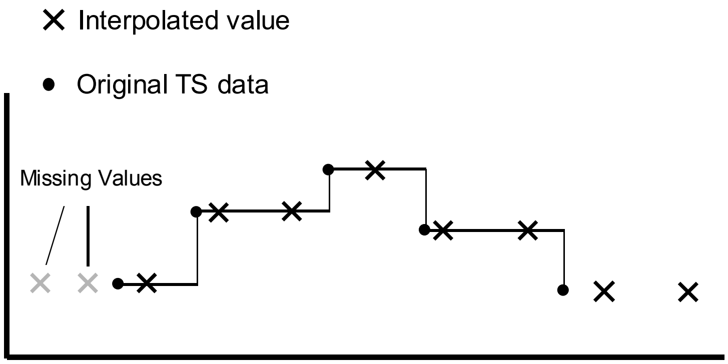


Figure 6.111 Interpolation of time series values using Period Constants function.

To generate a new time series of period constants:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Period Constants** operator.
2. From the **Selected Data Set** list, select the *primary* time series data set. This data set has the *values* for the interpolation.
3. From the **Data Set for Time Pattern** list, select the *second* time series data set. This data set has the *times* for the interpolation.
4. Click **Compute**.

6.6.10 Multiple Linear Regression

The **Multiple Linear Regression** function computes the multiple linear regression coefficients between a primary time series data set and a collection of independent time series data sets, and stores the regression coefficients in a new paired data set. This paired data set may be used with the Apply Multiple Linear Regression function to derive a new estimated time series (see section 6.6.11)

For the general linear regression equation, a dependent variable, Y, may be computed from a set independent variables, Xn:

Y = B0 + B1*X1 + B2*X2 + B3*X3

where Bn are the linear regression coefficients.

For time series data, an estimate of the primary time series values may be computed from a set of independent time series data sets using regression coefficients such that:

TsEstimate(t) = B0 + B1*TS1(t) + B2*TS2(t) + … + Bn*TSn(t)

where Bn are the set of regression coefficients and TSn are the time series data sets.

All the time series data sets must be regular interval and have the same time interval. The function filters the data to determine the time period common to all data sets and uses only those points in the regression analysis. For any given time, if a value is missing in any time series, no data for that time is processed. Optional minimum and maximum limits can be set to exclude values in the primary time series which fall outside a specified range.

To compute the set of multiple linear regression coefficients:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Multiple Linear Regression** operator.
2. From the **Selected Data Set** list, select primary time series data set . If the selected data set is not a time series data set, the remaining input boxes on the screen are unavailable.
3. From the **Independent Time Series Data Sets** list, select one or more of the independent time series data sets (by clicking, control-click or shift-click).
4. Click **Compute** to derive new a paired data set containing the linear regression coefficients.

6.6.11 Apply Multiple Linear Regression

The **Apply Multiple Linear Regression** function applies the multiple linear regression coefficients computed with the Multiple Linear Regression function (see section 6.6.10). The coefficients, stored in a paired data set, are applied to a collection of independent time series data sets to derive a new estimated time series data set.

For time series data, an estimate of the primary time series values may be computed from a set of independent time series data sets using regression coefficients such that:

Y = B0 + B1*X1 + B2*X2 + B3*X3

where Bn are the linear regression coefficients.

For time series data, an estimate of the original time series values may be computed from a set of independent time series data using regression coefficients such that:

TsEstimate(t) = B0 + B1*TS1(t) + B2*TS2(t) + … + Bn*TSn(t)

where Bn are the set of regression coefficients and TSn are the time series data sets.

The number regression coefficients in the paired data set must be one more than the number of independent time series data sets. The collection of selected time series data sets should be in the same order as when the regression coefficients were computed. The displayed list of time series is sorted alphabetically in the dialog. The user should be aware if the names of the time series data sets used for coefficient generation are substantially different from the data names used in this function.

All the time series data sets must be regular interval and have the same time interval. The function filters the data to determine the time period common to all time series data sets and uses only those points in the regression analysis. For any given time, if a value is missing in any time series, the value in the resultant time series is set to missing. You can also set optional minimum and maximum value limits. Computed values in the resultant time series, which fall outside the specified range, are set to missing.

Names, parameter type and unit labels for the resultant time series data set are taken from the first time series data set. The "F part" (Version) in the new data set is set to "COMPUTED."

To apply a set of multiple linear regression coefficients to derive a new time series data set:

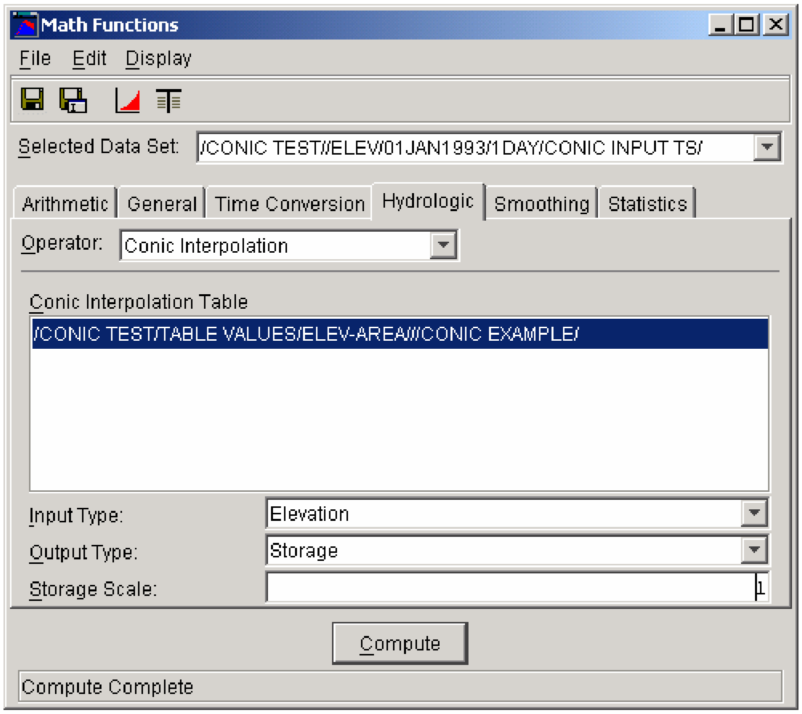
1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Apply Multiple Linear Regression** operator.
2. From the **Selected Data Set** list, select the paired data set containing the regression coefficients . If the selected data set is not a paired data set, the remaining input boxes on the screen are unavailable.
3. From the **Independent Time Series Data Sets** list, select one or more of the independent time series data sets. (You can select a single data set by clicking on it; select multiple contiguous data sets using **Shift+Click**, or select multiple, non-contiguous data sets using **Ctrl+Click**.)
4. Click **Compute**.

6.6.12 Conic Interpolation

The **Conic Interpolation** function transforms values in a time series data set by conic interpolation using an elevation-area table stored in a paired data set. The first value pair in the paired data set contains the initial conic depth and the initial storage volume at the first elevation (given in the next value pair). If the initial conic depth is missing, one is computed by the function. Values in the elevation-area table are stored in ascending order.

The Conic Interpolation function can interpolate a time series of elevation to derive a time series of storage or area. The function can interpolate a time series of storage to derive a time series of elevation or area. If the output time series is elevation, the output units are assigned from the x-units of the paired data set. If the output time series is area, the output units are assigned from the y-units of the paired data set. If the output time series is storage, the output units are undefined.

Figure 6.112 shows the appearance of the Math Functions screen for the Conic Interpolation function. The Conic Interpolation function is accessed from the **Hydrologic** tab of the Math Functions screen.



Selected Data Set: /CONIC TEST//ELEV/01JAN1993/1DAY/CONIC INPUT TS/

Operator: Conic Interpolation

Conic Interpolation Table

/CONIC TEST/TABLE VALUES/ELEV-AREA//CONIC EXAMPLE/

Input Type: Elevation

Output Type: Storage

Storage Scale: 1

Compute Complete

Figure 6.112 Screen for Conic Interpolation function

To perform the conic interpolation of a time series data set of elevation or storage:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Conic Interpolation** operator.
2. From the **Selected Data Set** list, select a time series data set. If the selected data set is not a time series, the remaining input boxes on the screen are unavailable.
3. From the **Conic Interpolation Table** list, select the paired data set containing the conic interpolation table. Only paired data sets will appear in this list, and only one data set at a time can be selected.
4. From the **Input Type** list, select the parameter type of the input time series data (Elevation or Storage).
5. From the **Output Type** list, select the desired output data type (Storage, Elevation or Area).
6. In the **Storage Scale** box, enter factor for the scale of the storage output values. By default this value is 1. The value is used to scale input (by multiplying) and output (by dividing) storage values. For example, if the area in the conic interpolation table is expressed in sq. ft., the storage scale could be set to 43560 to convert the storage output to acre-ft.
7. Click **Compute** to interpolate the selected time series data set.

6.6.13 Polynomial

The **Polynomial** function computes a polynomial transform of a regular or irregular interval time series data using the polynomial coefficients in a paired data set. Missing values in the input time series data remain missing in the resultant time series data set.

A new time series can be computed from an existing time series with the polynomial expression,

$$TS2\ (t) = B1^{*}\ TS1(t) + B2^{*}\ TS1(t)^{2} + \ldots + Bn^{*}\ TS1(t)^{n}$$

where Bn are the polynomial coefficients for term "n."

Values for the polynomial coefficients are stored in the x-values of a paired data set. Before the above equation is applied, values in the input time series are adjusted by subtracting off the paired data "datum" value if defined. The x-units and parameter type from the paired data set are applied to the resultant time series data set.

To compute the polynomial transform of a selected time series data set:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Polynomial** operator.
2. From the **Selected Data Set** list, select a time series data set. If the selected data set is not a time series, the remaining input boxes on the screen will be unavailable.
3. From the **Polynomial Coefficients** list, select the paired data set containing the polynomial coefficients. Only paired data sets will appear in this list, and you may select only one data set at a time.
4. Click **Compute** to perform the polynomial transform of the selected time series data set.

6.6.14 Polynomial with Integral

The **Polynomial with Integral** function computes a polynomial transform with integral of a regular or irregular interval time series data using the polynomial coefficients in a paired data set. Missing values in the input time series data remain missing in the resultant time series data set.

The polynomial transform coefficients are integrated, with the new time series values computed from an existing time series by the expression,

$$TS2\ (t) = B1^{*}\ TS1(t)^{2}/2\ \ + B2^{*}TS1(t)^{3}/3 + \ldots + Bn^{*}\ TS1(t)^{n+1}/(n+1)$$

where Bn are the polynomial coefficients for term "n."

Values for the polynomial coefficients are stored in the x-values of a paired data set. Before the above equation is applied, values in the input time series are adjusted by subtracting off the paired data "datum" value if defined. The x-units and parameter type from the paired data set are applied to the resultant time series data set.

To compute the polynomial transform with integral of a selected time series data set:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Polynomial with Integral** operator.
2. From the **Selected Data Set** list, select a time series data set. If the selected data set is not a time series, the remaining input boxes on the screen will be unavailable.
3. From the **Polynomial Coefficients** list, select the paired data set containing the polynomial coefficients. Only paired data sets will appear in this list, and you may select only one data set at a time.
4. Click **Compute**.

6.6.15 Flow Accumulator Gage Processor

The **Flow Accumulator Gage Processor** function computes the time series period-average flows from a flow accumulator type gage. Accumulator gage data consists of time series data sets of accumulated flow and counts. The two input time series data sets must match times exactly.

A new time series data set is derived from the time series of flow and counts by:

TsNew(t) = (TsAccFlow(t) - TsAccFlow(t-1)) /
(TsCount(t) - TsCount(t-1))

where TsAccFlow is the gage accumulated flow time series and TsCount is the gage time series of counts.

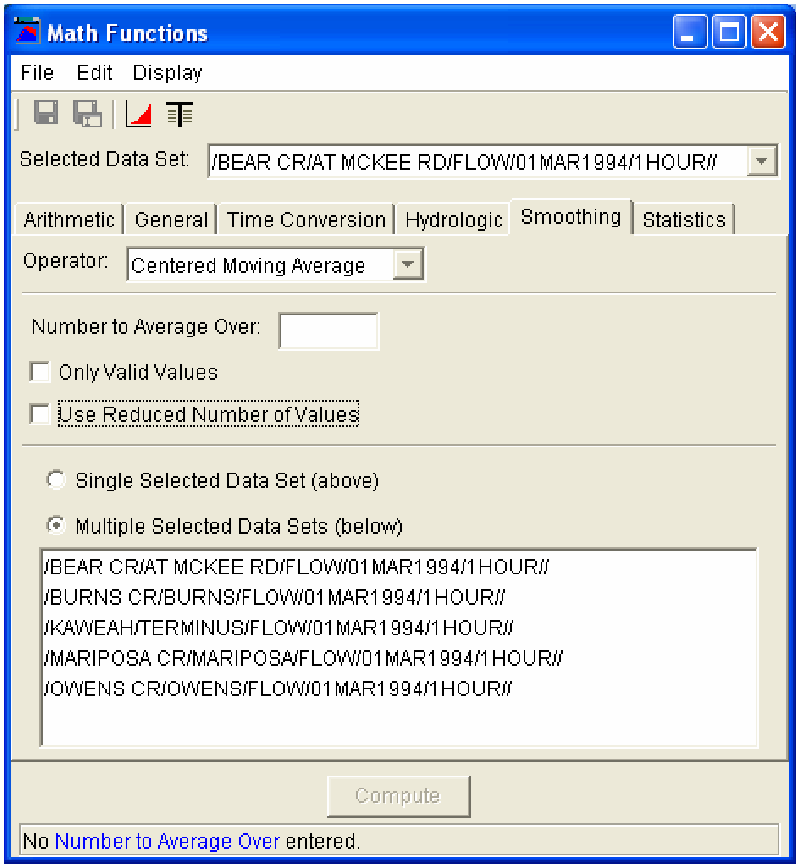
In the above equation, if TsAccFlow(t), TsAccFlow(t-1), TsCount(t) or TsCount(t-1) are missing, TsNew(t) is set to missing. The new time series is assigned the data type "PER-AVER."

To process the flow accumulator type data:

1. Choose the **Hydrologic** tab of the Math Functions screen and select the **Flow Accumulator Gage Processor** operator.
2. From the **Selected Data Set** list, select a time series data set.
3. From the **Data Set Containing Time Series of Counts** list, select the time series data set of counts.
4. Click **Compute**.

6.7 Smoothing Functions

The **Smoothing** tab (Figure 6.113) contains the functions Centered Moving Average, Olympic Smoothing Average, and Forward Moving Average.



Selected Data Set: /BEAR CR/AT MCKEE RD/FLOW/01MAR1994/1HOUR//

Operator: Centered Moving Average

Number to Average Over:

□ Only Valid Values

□ Use Reduced Number of Values

○ Single Selected Data Set (above)

● Multiple Selected Data Sets (below)

/BEAR CR/AT MCKEE RD/FLOW/01MAR1994/1HOUR//
/BURNS CR/BURNS/FLOW/01MAR1994/1HOUR//
/KAWEAH/TERMINUS/FLOW/01MAR1994/1HOUR//
/MARIPOSA CR/MARIPOSA/FLOW/01MAR1994/1HOUR//
/OWENS CR/OWENS/FLOW/01MAR1994/1HOUR//

No Number to Average Over entered.

Figure 6.113 Math Functions--Smoothing Tab

6.7.1 Centered Moving Average

The **Centered Moving Average** function computes a centered moving average of "NAVG" values for regular or irregular interval time series data. The number of values to average over ("NAVG") must be an odd integer greater than 2.

The **Only Valid Values** option pertains to when the averaging interval contains missing values. If checked, the option sets the value in the resultant data set to missing. If unselected, the option computes a smoothed value from the valid values in the interval.

The **Use Reduced Number of Values** option pertains to the first and last NAVG/2 values in the resultant data. If selected, the option computes smoothed values at the beginning and ending of the data set from a reduced number of values of the original data set. If unselected, the option sets the first and last NAVG/2 values in the resultant data to missing.

To compute the centered moving average for time series data:

1. Choose the **Smoothing** tab of the Math Functions screen and select the **Centered Moving Average** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Number to Average Over** box, enter a value to set the number of values for the moving average. This number must be an odd number greater than two (2).
4. Select the **Only Valid Values** box to compute a smoothed value only if all the values in the averaging interval are valid (no missing values). If there are one or more missing values, the value in the resultant time series data is then set to missing. If the box is not checked, the value in the resultant time series data is computed using the remaining valid values in the averaging interval.
5. Select the **Use Reduced Number of Values** box to compute a moving average from a reduced number of time series values near the beginning and ending of a time series. At these locations, the moving average interval becomes truncated and there are not "Number to Average Over" values for averaging. If unchecked, values in the resultant time series data at these locations will be set to missing.
6. Click **Compute** to perform a centered moving average smoothing of the selected data.

6.7.2 Olympic Smoothing Average

The **Olympic Smoothing Average** function uses the same smoothing scheme as the Centered Moving Average function (see section 6.7.1), except the minimum and maximum values in the averaging interval are excluded from the computation. The input time series may be regular or irregular interval. The number of values to average over ("NAVG") must be an odd integer greater than 2.

The **Only Valid Values** option pertains to when the averaging interval contains missing values. If checked, the option sets the value in the resultant data to missing. If unselected, the option computes a smoothed value from the valid values in the interval.

The **Use Reduced Number of Values** option pertains to the first and last NAVG/2 values in the resultant data. If selected, the option computes smoothed values at the beginning and ending of the data from a reduced number of values of the original data. If unselected, the option sets the first and last NAVG/2 values in the resultant data to missing.

The span of the averaging interval, "NAVG," must be an odd integer greater than two.

Under two conditions there are not NAVG valid values over the averaging interval: if there are missing values in the averaging interval, or if the smoothed point is within NAVG/2 values of the beginning or ending of the time series.

For the first condition, if the averaging interval contains any missing values, the option is to set the value in the resultant data set to missing, or compute a smoothed value from the remaining valid values in the interval.

For the second condition, the option is to set the first and last NAVG/2 values in the resultant data set to missing, or compute smoothed values at the beginning and ending of the data set from a reduced number of values.

To compute the Olympic smoothing average for a time series data set:

1. Choose the **Smoothing** tab of the Math Functions screen and select the **Olympic Smoothing Average** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Number to Average Over** box, enter a value to set the number of values for the averaging interval. This number must be an odd integer greater than 2.
4. Select the **Only Valid Values** box to compute a smoothed value only if all the values in the averaging interval are valid (no missing values). If there are one or more missing values, the value in the resultant time series data is then set to missing. If the box is not checked, the value in the resultant time series data is computed using the remaining valid values in the averaging interval.
5. Select the **Use Reduced Number of Values** box to compute a moving average from a reduced number of time series values near the beginning and ending of a time series. At these locations, the moving average interval becomes truncated and there are not "Number to Average Over" values for averaging. If unchecked, values in the resultant time series data at these locations will be set to missing.
6. Click **Compute** to perform an Olympic smoothing of the selected data.

.

6.7.3 Forward Moving Average

The **Forward Moving Average** function computes a moving average of the last "NAVG" values for regular or irregular interval time series data. The number of values for averaging, "NAVG," must be greater than 2. The first NAVG-1 values in the resultant time series are set to missing.

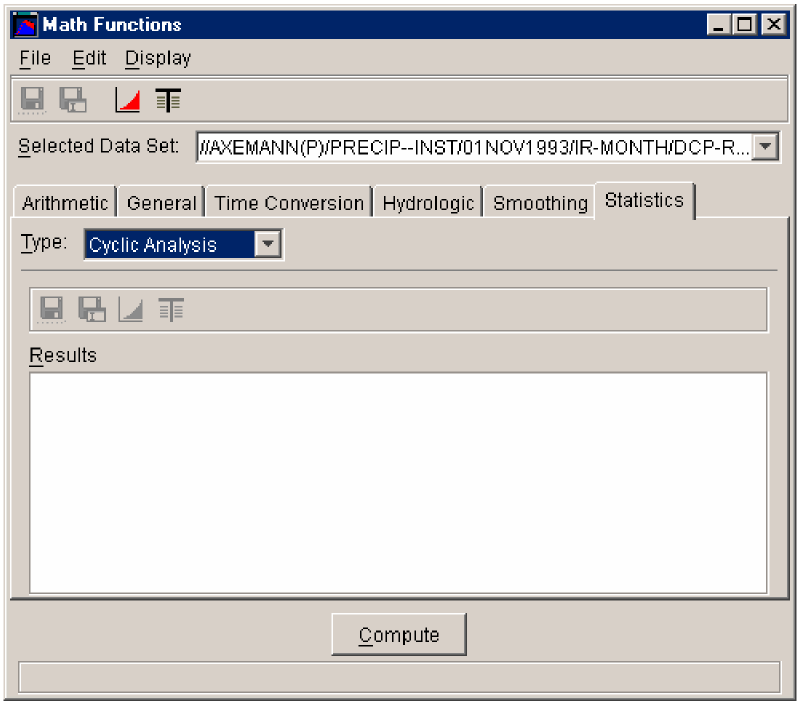
If the averaging interval contains a missing value, the smoothed value is computed from the remaining valid values in the interval. However, if there are less than 2 valid values in the interval, the value in the resultant data is set to missing.

To compute the forward moving average for time series data:

1. Choose the **Smoothing** tab of the Math Functions screen and select the **Forward Moving Average** operator.
2. Select a data set to apply the function from the **Selected Data Set** pull-down list at the top of the screen or multiple data sets from the list in the lower portion of the screen. If you include the data set selected in the top **Selected Data Set**, the operation on that set will only be done once.
3. In the **Number to Average Over** box, enter the number of values in the averaging interval. This number must be greater than two (2).
4. Click **Compute**.

6.8 Statistics Functions

The **Statistics** tab (Figure 6.114) contains the functions Basic (statistics), Linear Regression, and Cyclic Analysis.



Selected Data Set: //AXEMANN(P)/PRECIP--INST/01NOV1993/IR-MONTH/DCP-R...

Type: Cyclic Analysis

Figure 6.114 Math Functions--Statistics Tab

6.8.1 Basic

The **Basic** function computes the basic statistical values for a regular or irregular interval time series data set. The statistical and informational values displayed are:

Number of valid values
Number of missing values
Last valid value
Last valid value date and time
Minimum value
Minimum value date and time
Mean value
Maximum value
Maximum value date and time
Accumulated value for the time series
Standard deviation
Skew coefficient
Data type ("INST-VAL," "INST-CUM," "PER-AVER," "PER-CUM")

Data Units ("ft", "cfs", etc.)

To compute the basic statistical parameters for a time series data set:

1. Choose the **Statistics** tab of the Math Functions screen and select the **Basic** type.
2. From the **Selected Data Set** list, select a time series data set.

The statistics are displayed for the time series data set once the data set is selected.

6.8.2 Linear Regression

The **Linear Regression** function computes the linear regression and other correlation coefficients between two time series data sets. Values in the primary time series data set and the second time series data set are matched by time to form data pairs for the correlation analysis. Missing values are ignored. Times for the two time series data sets must match exactly. The data sets may be either regular or irregular interval time series data.

The correlations statistics computed by the function are:

Number of valid values
Regression constant
Regression coefficient
Determination coefficient
Standard error of regression
Determination coefficient adjusted for degrees of freedom
Standard error adjusted for degrees of freedom

The primary time series data set forms the values of the independent variable (x-values), while values of the second time series data set comprise the dependent variable (y-values). The linear regression coefficients express how values in the second data set can be derived from values in the primary data set:

TS2(t) = a + b * TS1(t)

where "a" is the regression constant and "b" the regression coefficient.

To compute the linear regression and correlation coefficients between two time series data sets:

1. Choose the **Statistics** tab of the Math Functions screen and select the **Linear Regression** type.
2. From the **Selected Data Set** list, select the primary time series data set for analysis. Time series values from this data set form the independent variable (x-values) of the correlation analysis.
3. From the **Dependent Data Set** list, select the second time series data set for analysis. Time series values from this data set will comprise the dependent variable (y-values) of the correlation analysis.

The correlation statistics are computed automatically once the two data sets are selected.

6.8.3 Cyclic Analysis

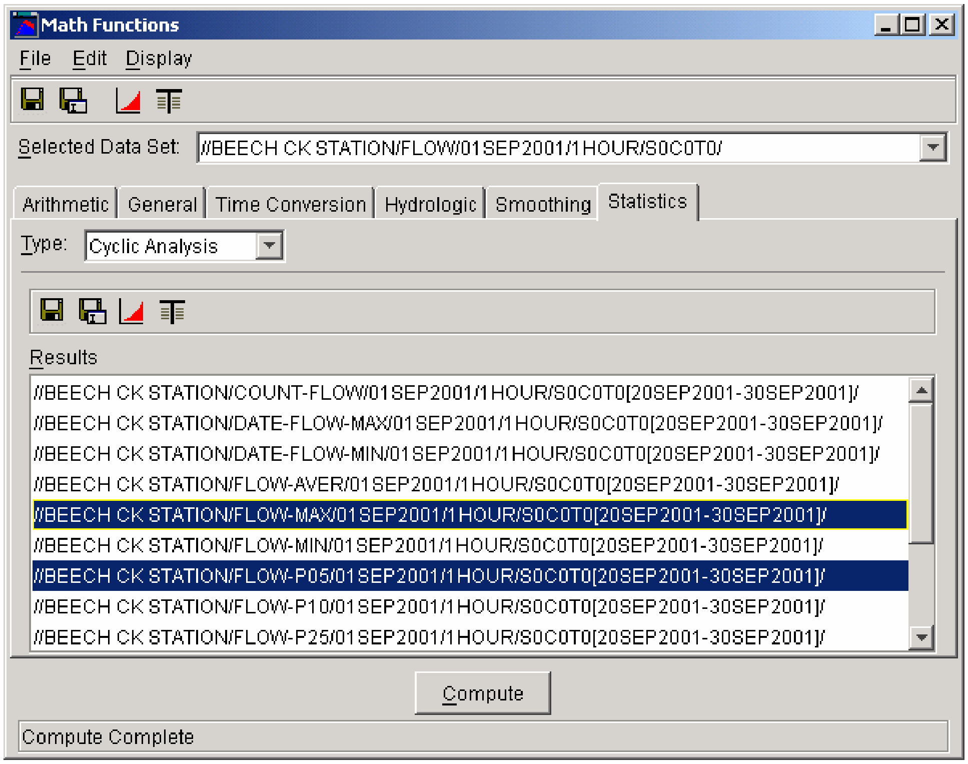
The **Cyclic Analysis** function derives a set of cyclic statistics from a regular interval time series data set. The time series data set must have a time interval of "1HOUR," "1DAY" or "1MONTH." The function sorts the time series values into statistical "bins" relevant to the time interval. Values for the 1HOUR interval data are sorted into 24 bins representing the hours of the day, 0100 to 2400. The 1DAY interval data is apportioned to 365 bins for the days of the year. The 1MONTH interval data is sorted into 12 bins for the months of the year.

The format of the resultant data sets is as a "pseudo" time series for the year 3000. For example, the cyclic analysis of one month of hourly interval data will produce pseudo time series data sets having 24 hourly values for the day January 1, 3000. If the statistical parameter is the "maximum" value, then the 24 values represent the maximum value occurring at that hour of the day in the original time series. The cyclic analysis of daily interval data will produce pseudo time series data sets having 365 daily values for the year 3000. The cyclic analysis of monthly interval data will result in pseudo time series data sets having 12 monthly values for the year 3000.

Fourteen pseudo time series data sets are derived by the cyclic analysis function for the following statistical parameters:

Number of values processed for each time interval
Maximum value
Time of maximum value
Minimum value
Time of minimum value
Average value
Probability exceedence percentiles for 5%, 10%, 25%, 50% (median value), 75%, 90%, and 95%
Standard deviation

Figure 6.115 shows the appearance of the Math Functions screen for the Cyclic Analysis function.



Selected Data Set: //BEECH CK STATION/FLOW/01SEP2001/1HOUR/S0C0T0/

Type: Cyclic Analysis

Results

```
//BEECH CK STATION/COUNT-FLOW/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/DATE-FLOW-MAX/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/DATE-FLOW-MIN/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/FLOW-AVER/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/FLOW-MAX/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/FLOW-MIN/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/FLOW-P05/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/FLOW-P10/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
//BEECH CK STATION/FLOW-P25/01SEP2001/1HOUR/S0C0T0[20SEP2001-30SEP2001]/
```

Compute Complete

Figure 6.115 Screen for Cyclic Analysis function

To compute the cyclic analysis of a time series data set:

1. Choose the **Statistics** tab of the Math Functions screen and select the **Cyclic Analysis** type.
2. From the **Selected Data Set** list, select a time series data set for cyclic analysis.
3. Click **Compute**.

Once the compute is performed, the resultant 14-pseudo time series data sets appear in **Results** list on the screen (Figure 6.115). One or more data sets in this list may be selected (by clicking, control-click or shift-click) for saving to file, plotting or tabulation by using the **Save** button, **SaveAs** button, the **Plot** button or the **Tabulate** button from the toolbar located immediately above the Results list.

Chapter 7

Scripting

Scripting provides a way to control the operation of HEC-DSSVue in a non-interactive way. The user can build and save scripts to be executed later – possibly on different data sets.

This chapter provides an introduction to scripting, describing the components of the user interface, scripting language and application program interface (API), and offering examples illustrating how to use the API.

7.1 Executing Scripts

HEC-DSSVue allows the execution of scripts in interactive and batch modes. Scripts are executed interactively by starting the HEC-DSSVue program and selecting the desired script from the Toolbar or the Script Selector from the **Utilities** menu. Scripts are executed in batch mode by starting the HEC-DSSVue program with a script file name as a parameter (e.g. `HecDssVue.bat c:\test\myScript.py`).

Interactive scripts are not passed any parameters upon script execution. In a script executed interactively the variable sys.argv is a list of length 1, with the only element set to the empty string (e.g. `sys.argv = ["″]`).

Scripts executed in batch mode may take parameters from the command line (e.g `HecDssVue.bat c:\test\myScript.py a b c`). In a script executed in batch mode the variable sys.argv is a list whose length is one greater than the number of parameters passed on the command line, with the first element set to the file name of the executing script and the remaining elements set to the parameters (e.g. `sys.argv = ["c:\\test\\myScript.py″, "a″, "b″, "c″]`).

7.1.1 Script Selector

The **Script Selector** (Figure 7.1) displays buttons for all the available scripts which have the "Display Script on Toolbar" box checked (see Section 7.2.2, " Editor Panel"). Buttons are displayed in alphabetical order.

To access the **Script Selector**, select the **Script Selector** command from the **Utilities** menu of HEC-DSSVue. Once the **Script Selector** is open, it will remain open until you close it.

When you press a button, the Jython script engine will execute the associated script.

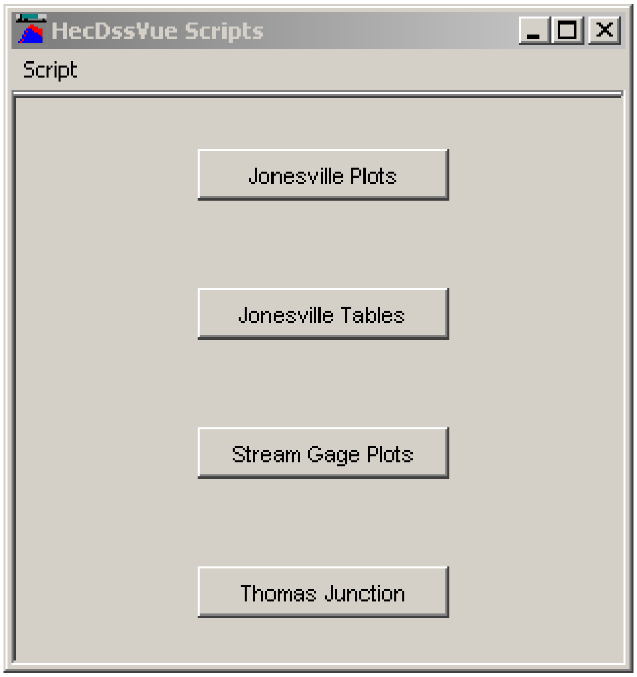
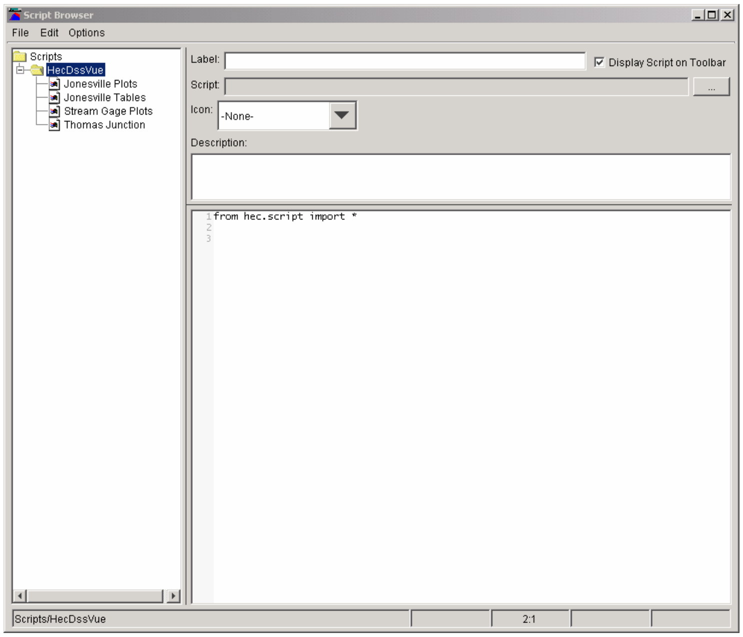


Figure 7.1 Script Selector

7.2 Script Browser

The **Script Browser**, shown in Figure 7.2, allows you to add, delete, and modify scripts.



☑ Display Script on Toolbar

Icon: -None-

```
from hec.script import *
```

Scripts/HecDssVue 2:1

Figure 7.2 Script Browser

You can access the **Script Browser** from HEC-DSSVue's **Utilities** menu by clicking **Script Browser**. Alternatively, from the shortcut menu of the **Script Selector**. In the Script Selector, right click on a button to access the shortcut menu, then select **Edit**. The Script Browser will open with that script selected and ready for editing.

Components of the **Script Browser** include the Menu Bar, the Editor Panel, and the Tree Hierarchy. The following sections describe these components.

7.2.1 Menu Bar

The **Menu Bar** (Figure 7.3) contains three primary menu items, **File** , **Edit**, and **Options**.

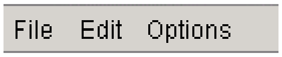


Figure 7.3 Menu Bar

File Menu Commands

New Creates a new script stored at the currently selected position. Available only when a folder node is the selected node in the scripts tree.

Open Edits the currently displayed script. Double clicking on the script also edits the currently displayed script. Available only when a script node is the selected node in the scripts tree.

Import Imports a file into the script browser. If the import is successful the browser is placed in edit mode. Available only when a folder node is the selected node in the scripts tree.

Save Saves the current script. Available only when a script is being edited.

Save As Saves the current script, allowing the user to change the label. Note that the name of the script file does not change. Available only when a script is being edited.

Delete Deletes the currently opened script. Prompts user for confirmation. Available only when a script node is the selected node in the scripts tree.

Test Executes the currently selected script.

Close Closes the Script Browser Window.

Edit Menu Commands

Cut Script Cuts the script at the currently selected tree node to the system clipboard. Available only when a script node is selected in the tree view.

Copy Script Copies the script at the the currently selected tree node to the system clipboard. Available only when a script node is selected in the tree view.

Paste Script Pastes the script in the system clipboard to the currently selected tree node. Available only when a folder node is selected in the tree view.

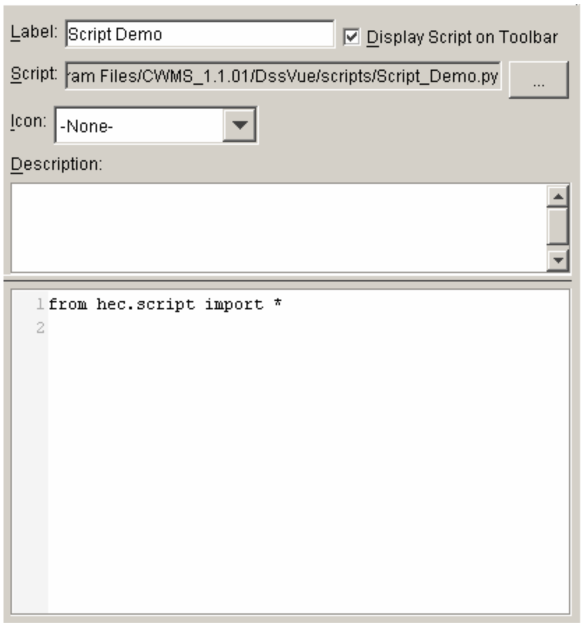
Option Menu Commands

Set Font Opens a dialog box that allows setting of the font used in the script text area. Fixed-space fonts such as "Courier New" and "Lucida Console" are recommended over proportional fonts.

Set Tab Size Opens a dialog box that allows setting the number of spaces that will be displayed in the script text area for each tab character.

7.2.2 Editor Panel

You can select and edit scripts in the **Editor Panel** of the Script Browser (Figure 7.4).



Label: Script Demo ☑ Display Script on Toolbar

Script: ram Files/CWMS_1.1.01/DssVue/scripts/Script_Demo.py

Icon: -None-

Description:

```
from hec.script import *

```

Figure 7.4 Editor Panel

The **Label** field allows you to specify the label displayed on a script's button in the Script Selector.

Script displays the name of the file in which the script is stored.

Display Script on Toolbar, when checked, enables the script to display in the **Script Selector** and the Toolbar. When you uncheck this option, the script will not display on the Script Selector or Toolbar.

The **Icon** field allows you to choose the Icon to display for the script's button. If you do not select an icon, the script name displays in the script's button.

The **Description** field allows you to add a description of the script. The first line of your description serves as a tooltip for the corresponding button on the Script Selector and Toolbar.

The **Script Text** field contains the script text itself and serves as an editing window for creating new scripts.

The script text field has a context menu that can be accessed by right-clicking in the script text field.

Script Text Field Context Menu Commands

Cut Copies the currently-selected script text to the system clipboard and removes it from the script.

Copy Copies the currently-selected script test to the system clipboard and leaves it in the current script.

Paste Copies text in the system clipboard into the script at the current cursor location.

Select All Selects all the text in the script.

Find Opens a dialog that allows the user to search for specific text in the script. If text is currently selected in the script, the dialog is initialized with this text.

Find Next Locates the next text in the script that matches the conditions of the most recently executed find command.

Find Previous Locates the previous text in the script that matches the conditions of the most recently executed find command.

Goto Line Opens a dialog that allows the user to cause the cursor to jump to the beginning of a specified line in the script text.

7.2.3 Tree Hierarchy

The **Tree Hierarchy** (Figure 7.5) uses a Windows Explorer-style tree structure to allow you to navigate folders in your directory structure and access scripts. By default, the scripts are stored in a "scripts" directory under the directory where HEC-DSSVue was installed.

The Tree Hierarchy also has a context menu that displays **Cut Script**, **Copy Script**, and **Edit Script** commands for script nodes and **New Script**, **Import Script**, and **Paste Script** for folder nodes. These commands cause the same actions as the File and Edit menu commands discussed above.

To access the context menu, point to a node in the "tree" and right-click with your mouse.

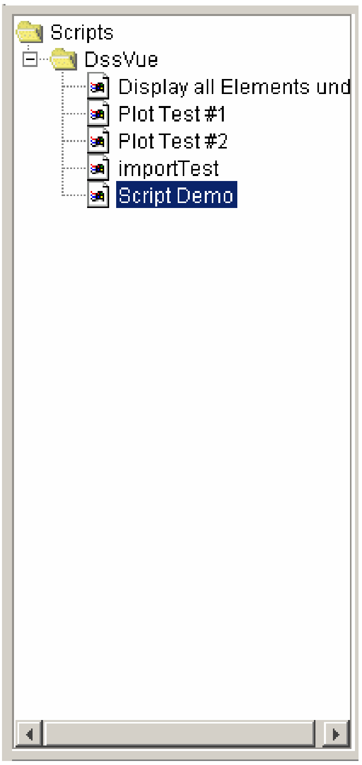


Figure 7.5 Tree Hierarchy

7.3 Scripting Basics

Scripting in HEC-DSSVue is accomplished using Jython, an implementation of the Python programming language designed specifically for integration with the Java programming language. More information about Jython can be found at the official Jython website – www.jython.org.

Python (of which Jython is an implementation) is an interpreted language with simple syntax and high-level data types. This section is not a comprehensive Python tutorial, but rather a simple primer to allow the creation of simple scripts. This primer does not cover defining classes in Python.

The official Python website - www.python.org - has links to online Python tutorials as well as programming books.

7.3.1 Outputting Text

Text information can be displayed in the console window using the print statement which has the syntax:

print [*item*[, *item*…]]

The *item*s are comma-separated and do not need to be of the same type. The print statement will insert a space between every specified item in the output.

Example 1: Outputting Text

```
print “Testing myFunction, i =”, i, “, x =”, x
```

7.3.2 Data Types

Python has integer, long integer, floating-point, imaginary number, and sequence and dictionary data types. Sequences are divided into mutable (or changeable) sequences called lists, immutable sequences called tuples. Strings are special tuples of characters that have their own syntax. Dictionaries are like sequences but are indexed by non-numeric values. In addition, Python also has a special type called None, which is used to indicate the absence of any value.

Python does not have a specific type for boolean (logical, or “true / false”) data. Tests, such as conditional expressions, which must evaluate to true or false are conducted such that the result is false if the expression evaluates to None, integer or floating-point zero, or an empty sequence. Any other result is true. Python statements that generate Boolean information (such as the **if** statement) generate integer 0 for false and integer 1 for true. This becomes an issue in Jython which allows calling Java functions and methods which expect a Java boolean for input or generate a Java boolean for output. Jython maps these boolean values to integer 0 or 1. Documentation for the HEC-DSSVue API uses the term Constants.TRUE (1), or Constants.FALSE (0), or sometimes the shorthand "0/1", for arguments (these are constants defined to 1 and 0, respectively, in hec.script), and "0/1" to specify that the return type is a Python integer, but its value is restricted to 0 or 1, corresponding to a Java boolean. The hec.script module supplies constants to use in these situations.

There are also situations regarding the HEC-DSSVue API where it is necessary or desirable to set a time-series value to "missing" or to test whether a time-series value is missing. The hec.script module also supplies a constant to use in these situations.

The currently-defined constants in the hec.script module are:

Constant	**Type**	**Represents**
Constants.TRUE	integer	true
Constants.FALSE	integer	false
Constants.UNDEFINED	floating-point	missing data value

It is recommended that these defined constants be used where applicable for portability and clarity.

Example 2: Variable Types

```
# set some integer values
i = 0
j = 1
k = -10998
m = Constants.TRUE

# set a long integer
n = 79228162514264337593543950336L

# set some floating-point values
x = 9.375
y = 6.023e23
z = -7.2e-3
t = Constants.UNDEFINED

# set some strings
string_1 = “abc”
string_2 = ‘xyz’
string_3 = “he said \“I won’t!\””
string_4 = ‘he said “I will not!”‘
string_5 = “””this is a
            multi-line string”””

# set a tuple – tuples are contained within ()
tuple_1 = (1, 2, “abc”, x, None)

# set a list – lists are contained within []
list_1 = [1, 2, “abc”, x, tuple_1]

# set a dictionary, using key : value syntax
# dictionaries are contained within {}
dict_1 = {“color” : “red”, “size” : 10, “list” : [1, 5, 8]}
```

Indexing into sequence types is done using [i] where i starts at 0 for the first element . Subsets of sequence types (called slices) are accessed using [i:j] where i is the first element in the subset and j is the element *after* the last element in the subset. If negative numbers are used to specify and index or slice, the index is applied to the *end* of the sequence, where [-1] specifies the last element, [-2] the next-to last and so on. If i is omitted in slice syntax it defaults to 0. If j is omitted in slice syntax it defaults to the length of the sequence, so `list_1[0:len(list_1)]` is the same as `list_1[:]`. Indexing into dictionaries is done using [x] where x is the key.

The number of elements in a sequence type or dictionary is returned by the **len()** function.

Example 3: Sequence Indexing and Slicing

```
string_4[3]              # 4th element
string_4[3:5]            # 4th & 5th elements
list_1[-1]               # last element
list_1[2:-1]             # 3rd through next-to-last element
list_1[2:len(list_1)]    # 3rd through last element (also list_1[2:])
dict_1["size"]           # value associated with "size" key
i = len(list_1)          # length of list_1
```

7.3.3 Variables

Python variable names consist of an upper- or lower-case letter or the "_" (underscore) character followed by an unlimited number of upper- or lower-case characters, digits or underscore characters.

Variables are assigned values by use of the "=" (equals) character. A sequence may be assigned to multiple variables using a single equals character. Variable names are case sensitive, so the name "startdate" is not the same name as "startDate".

Example 4: Assigning Values to Variables

```
i = 0
j = 1
k = -10998
string_1 = "abc"
i, j, k, string_1 = 0, 1, -10998, "abc"
```

7.3.4 Operators

Each of the following operators can be used in the form a = b operator c. Each can also be used as an assignment operator in the form a operator= b (e.g. a += 1, x **= 2).

+	arithmetic addition
-	negation or arithmetic subtraction
*	arithmetic multiplication
/	arithmetic division
**	arithmetic power
%	arithmetic modulo
&	bit-wise and
|	bit-wise or
~	bit-wise not
^	bit-wise xor (exclusive or)
<<	bit-wise left shift
>>	bit-wise right shift

Each of the following operators returns 0 (false) or 1 (true) and can be used in conditional expressions as discussed in Section 0.

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to
==	equal to

7.3.5 Comments

Python uses the "#" (hash) character to indicate comments. Everything from the "#" character to the end of the line is ignored by the interpreter. Comments may not be placed after a program line continuation ("\") character on the same input line.

7.3.6 Program Lines

Unless otherwise indicated, every input line corresponds to one logical program statement. Two or more statements can be combined on line input line by inserting the ";" (semicolon) character between adjacent statements. A single statement may be continued across multiple input lines by ending each line with the "\" (back slash) character. Comments may not be placed after a program line continuation ("\") character on the same input line.

Example 5: Input vs. Program Lines

```
# multiple statements per line
r = 1; pi = 3.1415927; a = pi * r ** 2

# multiple lines per statement
a = \
  pi * \
  r ** 2
```

Input lines are grouped according to their function. Input lines forming the body of a conditional, loop, exception handler, or function or class definition must be grouped together. Input lines not in any of the construct comprise their own group. In Python, grouping of input lines is indicated by indentation. All lines of a group must be indented the same number of spaces. A horizontal tab character counts as 8 spaces on most systems. In some Python documentation, a group of input lines is called a *suite*.

Example 6: Input Line Grouping

```
# this is the main script group
dist = x2 – x1
if dist > 100.:
   # this is the "if" conditional group
   y = dist / 2.
   z = y ** 2.
else :
   # this is the "else" conditional group
   y = dist.
   z = y ** 2. / 1.5
# back to main script group
q = y * z
```

7.3.7 Conditional Expressions

Conditional expressions have the form:

```
if [not] condition :
  if-group
[elif [not] condition :
  elif-group]
[else :
  else-group]
```

The ":" (colon) character must be placed after each condition.

The condition in each test is an expression built from one or more simple conditions using the form:

```
simple-condition ( and | or ) [not] simple-condition
```

Parentheses can be used to group conditions.

The simple-condition in each expression is either an expression using one of the conditional operators mentioned in Section 0 or is of the form:

```
item [not] in sequence
```

Example 7: Conditional Expressions

```
if (x < y or y >= z) and string_1.index("debug") != -1 :
   # do something
   …
elif z not in value_list or (x < z * 2.5) :
   # do something different
   …
else :
   # do something else
```

If the statement group to be processed upon a condition is a single statement, that statement may follow the condition on the same line (after the colon character).

Example 8: Simple Conditional Expressions

```
if x1 < x2 : xMax = x2
else       : xMax = x1
```

7.3.8 Looping

Python supports conditional looping and iterative looping. For each type, the body of the loop (the loop-group) can contain **break** and/or **continue** statements.

The **break** statement immediately halts execution of the loop-group and transfers control to the statement immediately following the loop.

The **continue** statement skips the remainder of the current iteration of the loop-group and continues with the next iteration of the loop-group.

7.3.8.1 Conditional Looping

Python supports conditional looping with the ***while*** statement, which has the form:

```
while condition :
   loop-group
```

Conditional looping executes the body of the loop (the loop-group) as long as the condition evaluates to true.

Example 9: Conditional Looping

```
# print the first 10 characters
string_1 = “this is a test string”
i = 0
while i < 10 :
   print string_1[i]
   i += 1
```

7.3.8.2 Iterative Looping

Python supports iterative looping with the ***for*** statement, which has the form:

```
for item in sequence :
   loop-group
[else :
   else-group]
```

Iterative looping executes the body of the loop (the loop-group) once for each element in *sequence*, first setting *item* to be that element. If the iteration proceeds to completion without being interrupted by a break statement the else-group will be executed, if specified.

The **range([*start*,] *stop*[, *increment*])** helper function generates a sequence of numbers from *start* (default = 0) to *stop*, incrementing by *increment* (default = 1). Thus range(4) generates the sequence (0, 1, 2, 3).

Example 10: Iterative Looping

```
# print the first 10 characters
string_1 = “this is a test string”
for i in range(10) :
    print string_1[i]

# print all the characters
string_1 = “this is a test string”
for i in range(len(string_1)) :
    print string_1[i]

# print all the characters (more Python-y)
string_1 = “this is a test string”
for c in string_1 :
    print c
```

7.3.9 Defining and Using Functions

In Python, functions are defined with the syntax:

```
def functionName([arguments]) :
   function-body
```

Function names follow the same naming convention as variable names specified in Section 7.3.2. The arguments are specified as a comma-delimited list of variable names that will be used within the function-body. These variables will be positionally assigned the values used in the function call. More complex methods of specifying function arguments are specified in Python tutorials and references listed at the official Python website (www.python.org).

A function must be defined in a Python program before it can be called. Therefore, function definitions must occur earlier in the program than calls to those functions.

A function may optionally return a value or sequence of values.

Example 11: Defining and Using Functions

```
def printString(stringToPrint) :
  “Prints a tag plus the supplied string”
  tag = “function printString : “
  print tag + stringToPrint

def addString(string_1, string_2) :
  “Concatenates 2 strings and returns the result”
  concatenatedString = string_1 + string_2
  return concatenatedString

testString = “this is a test”
printString(testString)
wholeString = addString(“part1:”, “part2”)
printString(wholeString)
printString(addString(“this is “, “another test”))
```

7.3.10 Modules, Functions and Methods

A function is a procedure which takes zero or more parameters, performs some action, optionally modifies one or more of the parameters and optionally returns a value or sequence of values.

A class is the definition of a type of object. All objects of that type (class) have the same definition and thus have the same attributes and behavior. Classes may define functions that apply to individual objects of that type. These functions are called methods.

An object is an instance of a class, which behaves in the way defined by the class, and has any methods defined by the class.

Python provides many functions and classes by default. In our examples we have used functions len() and range() which Python provides by default. We have also used the classes list and string, which Python also provides by default. We didn’t use any methods of the class list, but we used the string method index() in the example in Section 0 (`string_1.index(“debug”) != -1`). It is important to note that the object method index() doesn’t apply to the string class in general, but to the specific string object string_1.

There are other functions and classes which Python does not provide by default. These functions and classes are grouped into modules according to their common purpose. Examples of modules are “os” for operating system functions and “socket” for socket-based network functions. Before any of the functions or classes in a module can be accessed, the module must be imported with the import statement, which has the syntax:

from *module* import *

Other methods of using the import statement are specified in Python tutorials and references listed at the official Python website (www.python.org). In the Jython implementation, Java packages can be imported as if they were Python modules, and the Java package java.lang is imported by default.

Example 12: Using a Function from an Imported Module

```
# use the getcwd() function in the os module to get
# the current working directory

from os import *
cwd = getcwd()
```

A module *does not* have to be imported in order to work with objects of a class defined in that module *if* that object was returned by a function or method already accessible. For example, the Python module "string" does not have to be imported to call methods of string objects, but *does* have to be imported to access string functions.

7.3.11 Handling Exceptions

Certain errors within a Python program can cause Python to raise an exception. An exception that is not handled by the program will cause the program to display error information in the console window and halt the program.

Python provides structured exception handling with the following constructs:

```
try :
  try-group
except :
  except-group
[else :
  else-group]


try :
  try-group
finally :
  finally-group
```

In the try-except-else construct, if an exception is raised during execution of the try-group control immediately transfers to the first line of the except-group. If no exception is raised during execution of the try-group control transfers to the first line of the else-group, if present. If there is no exception raised and no else-group is specified, the control transfers to the first line after the except-group.

In the try-finally construct, control is transferred to the first line of the finally-group when either an exception is raised during the execution of the try-group or the try-group completes without an exception.

The two constructs cannot be combined into a try-except-finally construct, but the same effect can be obtained by making a try-except-else construct the try-group of a try-finally construct.

Example 13: Exception Handling

```
try :
   try :
      string_1.find(substring) # may raise an exception
   except :
      print substring + " is not in " + string_1
      # do some stuff that might raise another exception
      …
   else :
      print substring + " is in " + string_1
      # do some stuff that might raise another exception
      …
finally :
   print "No matter what, we get here!"
```

More exception handling information, including filtering on specific types of exceptions, exception handler chains, and raising exceptions, is provided in Python tutorials and references listed at the official Python website (www.python.org).

7.4 Displaying Messages

It is often useful to display messages to inform the user that something has occurred, to have the user answer a Yes/No question, or offer debugging information to help determine why a script isn't working as expected. Text information can be displayed in the console window as described in Section 7.3.1, "Outputting Text."

7.4.1 Displaying Message Dialogs

The MessageBox class in the hec.script module, and the MessageDialog class in the MessageDialog module have several functions used to display messages in message box dialogs. The dialogs can be one of four different types: Error, Warning, Informational or Plain. The difference between the MessageBox functions and their MessageDialog counterparts is the the MessageBox versions do not allow interaction with any object other than the message box itself, while the MessageDialog versions allow the user to interact with the HecDssVue program window, as well as any displayed tables or plots while the message box is displayed. If MessageDialog functions are used in a script that is to be executed other than from the command line, they must not be used in the main script thread.

MessageBox and MessageDialog functions with multiple buttons return a string containing the text of the button that was selected to dismiss the message box.

Note: Do not use the MessageBox or MessageDialog functions in a script that is to run unattended since these functions cause scripts to pause for user interaction.

Table 7.1 describes MessageBox and MesssageDialog functions.

Table 7.1 – MessageBox and MessageDialog Functions

Function	Returns	Comments
showError(string message, string title)	None	Display an error dialog to you with the message and title
showWarning(string message, string title)	None	Display a warning dialog to you with the message and title
showInformation(string message, string title)	None	Display a Informational dialog to you with the message and title
showPlain(string message, string title)	None	Display a plain dialog to you with the message and title

Function	Returns	Comments
showYesNo(string message, string title)	string	Display a Yes/No dialog to you with the message and title
showYesNoCancel(string message, string title)	string	Display a Yes/No/Cancel dialog to you with the message and title
showOkCancel(string message, string title)	string	Display a Ok/Cancel dialog to you with the message and title

Example 14: Display Error Dialog with MessageBox class

```
from hec.script import *

MessageBox.showError("An Error Occurred", "Error")
```

Example 15: Display OK/Cancel Dialog with MessageDialog class

```
from hec.script import *
import MessageDialog
import thread

def main() :
   ok=MessageDialog.showOkCancel("Continue with Operation", "Confirm")

thread.start_new_thread(main, ())
```

7.5 Accessing the Main Program Window

Situations arise that necessitate accessing the main HEC-DSSVue program window. Examples of such situations are:

- Having a script determine whether it is running in an interactive mode or without a graphical display.
- Having a script gather information about interactively-selected pathnames and time windows for automated processing.
- Having a script launch the interactive graphical editor.

The ListSelection class in the hec.dssgui module facilitates these activities. An import statement of the form "from hec.dssgui import ListSelection" is necessary to use the ListSelection class.

If the script is not running in a mode without a graphical display (i.e. the HEC-DSSVue was launched with the script as a parameter) then a main program window will not exist. If the script needs to have the user utilize the graphical editor, the script will need to create a main program window for the duration of the graphical edit session.

7.5.1 ListSelection Class

Table 7.2 lists the static functions of the ListSelection class.

Table 7.2 – ListSelection Static Functions

Function	Returns	Description
createMainWindow()	ListSelection	Returns a new ListSelection object. This should not be called unless the script is not running in interactive mode.
getMainWindow()	ListSelection	Returns the main program window (ListSelection object) of the script. Returns None if the script is not running in interactive mode.

ListSelection objects can be used to gather information about pathnames and time windows that the user selected interactively before launching the script. ListSelection objects can also be used to launch the interactive graphical editor. Table 7.3 lists the ListSelection methods

Table 7.3 – ListSelection Methods

Method	Returns	Description
getLocation()	Point	Returns the location of the dialog in screen coordinates.[1]
getSelectedPathnames()	list of DataReference objects	Returns a list of DataReference objects that represent the interactive pathname and time window selections.
getSize()	Dimension	Returns the dimensions of the dialog in screen coordinates.
graphicalEdit()	None	Launches the graphical editor for the intereactively-selected pathnames and time windows.
graphicalEdit(TimeSeriesContainer tsc)	None	Launches the graphical editor for the specified TimeSeriesContainer.
graphicalEdit(list tscList)	None	Launches the graphical editor for all TimeSeriesContainers in the list.
isVisible()	0/1	Returns whether the dialog is currently visible on the screen.
setLocation(integer x, integer y)	None	Sets the location of the dialog in screen coordinates.[1]
setSize(integer width, integer height)	None	Sets the size of the dialog in screen coordinates.
setVisible(0/1 visible)	None	Sets whether the dialog is visible on the screen.

[1] The coordinate system used is a graphics coordinate system, where X increases to the right and Y increases downward from the origin (0,0) which is located in the top left corner of the display. Locations set or retrieved refer to the top left corner of the dialog in reference to this coordinate system.

7.5.2 DataReference Class

The getSelectedPathnames() ListSelection method returns a list of DataReference objects that represent the interactive selections. Each individual selection may have a different DSS filename, pathname, and time window than other selections. Table 7.4 lists the methods of the DataReference class.

Table 7.4 – DataReference Methods

Method	Returns	Description
getFilename()	string	Returns the name of the DSS file from which this selection was made.
getPathname()	string	Returns the pathname for this selection
getTimeWindow(HecTime startTime, HecTime endTime)	0/1	Sets the startTime and endTime parameters with the start time and end time of the selection, respectively. Returns Constants.TRUE if a time window is defined for the selection. Returns Constants.FALSE otherwise.
hasTimeWindow()	0/1	Returns Constants.TRUE if a time window is defined for the selection. Returns Constants.FALSE otherwise.

7.6 Reading and Writing to HEC-DSS Files

Reading or writing a data set from a DSS file involves using functions and methods from 3 classes in the hec.hecmath module: DSS, DSSFile and HecMath. The DSS class is used to get a DSSFile object which represents a DSS File. The DSSFile object is then used to get individual data sets out of the DSS File by returning a HecMath object.

7.6.1 DSS Class

```
DSS.open(string filename)
DSS.open(string filename, string timeWindow)
DSS.open(string filename, string startTime, string endTime)
```

The DSS class is used to gain access to a HEC-DSS File, as illustrated in Example 16.

Example 16: Opening a DSS File

```
theFile = DSS.open("MyFile.dss")
or
theFile = DSS.open("MyFile.dss", "T-14D T")
or
theFile = DSS.open("MyFile.dss", "01Jan2002,1300", "02Jan2002,1300")
```

7.6.2 DSSFile Class

DSSFile objects are used to read and write data sets in a DSS file, and to retrieve cataloged pathnames in a DSS file. Table 7.5 describes DSSFile methods

Table 7.5 - DSSFile Methods

Method	Returns	Description
close()	None	Close the DSS file.
getCatalogedPathnames()	list of strings	Returns a list of all cataloged pathnames without generating a new catalog.
getCatalogedPathnames(0/1 forceNew)	list of strings	Returns a list of all cataloged pathnames, optionally generating a new catalog first.
getCatalogedPathnames(string pattern)	list of strings	Returns a list of all cataloged pathnames that match the specified pattern without generating a new catalog.

Method	Returns	Description
getCatalogedPathnames(string pattern, 0/1 forceNew)	list of strings	Returns a list of all cataloged pathnames that match the specified pattern, optionally generating a new catalog first.
read(string pathname)[1]	HecMath	Return an HecMath object that holds the data set specified by pathname.
read(string pathname, string timeWindow) [1]	HecMath	Return an HecMath object that holds the data set specified by pathname with the specified time window.
read(string pathname, string startTime, string endTime) [1]	HecMath	Return an HecMath object that holds the data set specified by pathname with the specified time window.
setTimeWindow(string timeWindow)	None	The default time window for this DSSFile.
setTimeWindow(string startTime, string endTime)	None	The default time window for this DSSFile.
setTrimMissing(0/1 trim)	None	Sets whether TimeSeriesMath objects retrieved via calls to read(…) will have missing data trimmed from the beginning and end of the time window. [2]
write(HecMath dataset) [1]	integer	Write the data set to the DSS file. A return value of zero indicates success.

[1] Currently, the read(…) and write(…) methods can operate only on time-series data, paired data and stream rating data represented by TimeSeriesMath, PairedDataMath, and StreamRatingMath objects, respectively. Other record types, such as text data and gridded data are not yet supported by these methods.

[2] By default, TimeSeriesMath objects retrieved via calls to read(…) contain data only between the first non-missing value and the last non-missing value within the specified time window. Calling setTrimMissing(Constants.FALSE) causes the data retrieved to include all data for the specified time window, including blocks of missing values at the beginning and end of the time window.

Example 17: Reading a DSS Data Set

```
from hec.hecmath import *

# open myFile.dss and read a data set
theFile = DSS.open("myFile.dss")
flowDataSet = theFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/")
theFile.close()
```

7.6.2.1 Pattern Strings

The pattern strings used with the getCatalogedPathnames(…) methods filter the list of returned pathnames in a user-specified manner. Pattern strings can be specified in two modes: pathname mode and pathname part mode. Both modes make use of pathname part filters:

- Pathname mode = */AFilter/BFilter/CFilter/DFilter/EFilter/FFilter/*
- Pathname part mode = [**A=***AFilter*] [**B=***BFilter*] [**C=***CFilter*] [**D=***DFilter*] [**E=***AFilter*] [**F=***FFilter*]

In pathname mode, filters must be supplied for all pathname parts. In pathame part mode, only those parts that will not match everything must be specified.

Filters are comprised of the following components:

- **Normal text characters.** These characters are interpreted as they appear. For example, a pattern string of "B=XYZ" specifies matching every pathname that has a B-part of "XYZ".
- **Special characters.** The special characters are comprised of the following list:
  - **'@'** or **'*'** (used interchangeably). This character can be specified as the first and/or last character of a filter and specifies matching a string of zero or more characters. For example, a pattern string of "B=XYZ@" specifies matching every pathname that as a B-part that begins with "XYZ". A pattern string of "B=*XYZ" specifies matching every pathname that has a B-part that ends with "XYZ". A pattern string of "B=*XYZ*" specifies matching every pathname that "XYZ" anywhere in the B-part. Note that the **@** or ***** character must be the first and/or last character of the filter (e.g. a pattern string of "B=ABC*XYZ" is invalid).
  - **'#'** or **'!'** (used interchangeably). This character must be specified as the first character of a filter and specifies matching of every string *except* the remainder of the filter (e.g. it negates the remainder of the filter). A pattern string of "B=!XYZ*" specifies matching every pathname that does *not* have a B-part that begins with "XYZ".
- **No character**. The absence of any character specifies an empty filter, which matches a blank pathname part. Both of the following pattern strings match every pathname that has a blank A-part:
  - "//*/*/*/*/*/"
  - "A="

Using getCatalogedPathnames(…) with a pattern string utilizes the pathname matching in the underlying DSS library. To accomplish more sophisticated pathname filtering, use one of the getCatalogedPathnames(…) methods in conjunction with python text parsing and matching utilities.

7.6.2.2 Time Windows

Dates and times used to specify time windows should not contain blank characters and should include the 4 digit year. An example date and time is "04MAR2003 1400". If a single string is used to specify the time window, the starting date and time must precede the ending date and time, for example *dssFile*.setTimeWindow ("04MAR2003 1400 06APR2004 0900"). A relative time may be used in the single string command, where the letter "T" represents the current time, and the days or hours can be subtracted or added to that. For example *dssFile*.setTimeWindow("T-14D T-2H") would specify the starting time as current time minus 14 days and the ending time as the current time minus 2 hours.

Time Windows (specified by starting time and ending time) effect how the DSSFile.read(…) method operates for the various data types.

- **Time-series data.** The time window supplied to DSS.open(…) or *dssFile*.setTimeWindow(…) specifies the default time window for all subsequent *dssFile*.read(…) operations. If no time window is supplied to DSS.open(…) then the default time window is undefined until *dssFile*.setTimeWindow(…) is called. If the default time window is undefined , then all *dssFile*.read(…) operations involving time-series data *must* specify a time window either implicitly via the D-part of the specified pathname or explicitly via the startTime and endTime parameters. The order of precedence for time windows is as follows:
    - **Explicit time window.** Specified in *dssFile*.read(pathname, startTime, endTime) or *dssFile*.read(pathname, timeWindow). The D-part of the pathname is ignored and may be empty.
    - **Default time window.** Specified in DSS.open(filename, startTime, endTime) or DSS.open(filename, timeWindow) or *dssFile*.setTimeWindow(startTime, endTime) or *dssFile*.setTimeWindow(timeWindow). The D-part of the pathname is ignored and may be empty. The default time window can be set to undefined by calling *dssFile*.setTimeWindow("", "").
    - **Implicit time window.** Specified as the D-part of the pathname supplied to *dssFile*.read(pathname) when the default time window is undefined. The D-part of the pathname must not be empty.
- **Paired data**. Time windows have no effect on reading paired data records.

- **Stream rating data**. Stream rating data are comprised of a time-series of individual rating records for a common location. The reading of stream rating data is not effected by default time windows. The implicit time window for reading stream rating data is the entire time span covered by the rating records. An explicit time window may be supplied by using the *dssFile*.read(pathname, startTime, endTime) method. If an explicit time window is specified, a (possible) subset of the rating records is retrieved that cover the specified time window. The explicit time window is interpreted as the time window containing all time-series data that is to be rated via the stream rating data. The set of rating records retrieved for an explicit time window meets the following criteria.
    - The earliest record retrieved is the latest rating that is on or before the start of the time window. If no such record exists, the earliest record in the rating time-series is retrieved.
    - The latest record retrieved is the earliest rating that is on or after the end of the time window. If no such record exists, the latest record in the rating time-series is retrieved.
    - All rating records between the earliest and latest retrieved records are also retrieved.

7.7 HecMath Class

The objects returned from the *dssFile*.read(…) methods and supplied to the *dssFile*.write(…) method are HecMath objects. There are currently three types of HecMath classes: TimeSeriesMath class, PairedDataMath class, and StreamRatingMath class which represent time-series data, paired data, and stream rating data, respectively.

An HecMath object can be created for writing to new DSS data by first creating a new DataContainer as discussed in Sections 7.8.1 (TimeSeriesContainer Class) and 7.8.2 (PairedDataContainer Class), and then calling the HecMath.createInstance() function with the DataContainer object as the only parameter (e.g. myTimeSeriesMath = HecMath.createInstance(myTimeSeriesContainer)). Table 7.6 lists the methods for HecMath Objects, which are described in Section 7.15

Table 7.6 – HecMath Methods

Method	Returns	Description Section
abs()	HecMath	7.15.1
accumulation()	TimeSeriesMath	7.15.2
add(floating-point constant)	HecMath	7.15.3
add(TimeSeriesMath dataset)	TimeSeriesMath	7.15.4
applyMultipleLinearRegression(string startTimeString, string endTimeString, sequence datasets, floating-point minLimit, floating-point maxLimit)	TimeSeriesMath	7.15.5
centeredMovingAverage(integer number, 0/1 onlyValid, 0/1 useReduced)	TimeSeriesMath	7.15.6
conicInterpolation(TimeSeriesMath dataset, string inputType, string outputType, floating-point storageFactor)	TimeSeriesMath	7.15.7
convertValuesToEnglishUnits()	HecMath	7.15.8
convertValuesToMetricUnits()	HecMath	7.15.9
correlationCoefficients(TimeSeriesMath dataset)	LinearRegression Statistics	7.15.10
cos()	HecMath	7.15.11
cyclicAnalysis()	sequence of TimeSeriesMath	7.15.12
decayingBasinWetnessParameter(TimeSeriesMath tsPrecip, floating-point decayRate)	TimeSeriesMath	7.15.13

Method	Returns	Description Section
divide(floating-point constant)	HecMath	7.15.14
divide(TimeSeriesMath dataset)	TimeSeriesMath	7.15.15
estimateForMissingPrecipValues(integer maxMissing)	TimeSeriesMath	7.15.16
estimateForMissingValues(integer maxMissing)	TimeSeriesMath	7.15.17
exponentiation(floating-point constant)	HecMath	7.15.18
extractTimeSeriesDataForTimeSpecification(string timeLevel, string range, 0/1 isInclusive, integer intervalWindow, 0/1 setAsIrregular)	TimeSeriesMath	7.15.19
flowAccumulatorGageProcessor(TimeSeriesMath dataset)	TimeSeriesMath	7.15.20
forwardMovingAverage(integer number)	TimeSeriesMath	7.15.21
generateDataPairs(TimeSeriesMath dataset, 0/1 sort)	PairedDataMath	7.15.22
generateRegularIntervalTimeSeries(string startTime, string endTime, string timeInterval, string timeOffset, floating-point initialValue)	TimeSeriesMath	7.15.23
getData()	DataContainer	7.15.24
getType()	string	7.15.25
getUnits()	string	7.15.26
interpolateDataAtRegularInterval(string timeInterval, string timeOffset)	TimeSeriesMath	7.15.27
inverse()	HecMath	7.15.28
isEnglish()	0/1	7.15.29
isMetric()	0/1	7.15.30
isMuskingumRoutingStable(integer subReachCount, floating-point muskingumK, floating-point muskingumX)	string	7.15.31
lastValidDate()	integer	7.15.32
lastValidValue()	floating-point	7.15.33
log()	HecMath	7.15.35
log10()	HecMath	7.15.36
max()	floating-point	7.15.37

Method	Returns	Description Section
maxDate()	integer	7.15.38
mean()	floating-point	7.15.39
mergePairedData(PairedDataMath dataset)	PairedDataMath	7.15.40
mergeTimeSeries(TimeSeriesMath dataset)	TimeSeriesMath	7.15.41
min()	floating-point	7.15.42
minDate()	integer	7.15.43
modifiedPulsRouting(TimeSeriesMath dataset, integer subReachCount, floating-point muskingumX)	TimeSeriesMath	7.15.44
multipleLinearRegression(sequence datasets, floating-point minLimit, floating-point maxLimit)	PairedDataMath	7.15.45
multiply(floating-point constant)	HecMath	7.15.46
multiply(TimeSeriesMath dataset)	TimeSeriesMath	7.15.47
muskingumRouting(integer subReachCount, floating-point muskingumK, floating-point muskingumX)	TimeSeriesMath	7.15.48
numberMissingValues()	integer	7.15.49
numberValidValues()	integer	7.15.50
olympicSmoothing(integer number, 0/1 onlyValid, 0/1 useReduced)	TimeSeriesMath	7.15.51
periodConstants(TimeSeriesMath dataset)	TimeSeriesMath	7.15.52
polynomialTransformation(TimeSeriesMath dataset)	TimeSeriesMath	7.15.53
polynomialTransformationWithIntegral(TimeSeriesMath dataset)	TimeSeriesMath	7.15.54
ratingTableInterpolation(TimeSeriesMath dataset)	TimeSeriesMath	7.15.55
reverseRatingTableInterpolation(TimeSeriesMath dataset)	TimeSeriesMath	7.15.56
round()	HecMath	7.15.57
roundOff(integer digits, integer place)	HecMath	7.15.58
screenWithForwardMovingAverage(integer number, floating-point changeLimit, 0/1 setMissing, string invalidQuality)	TimeSeriesMath	7.15.59

Method	Returns	Description Section
screenWithMaxMin(floating-point min, floating-point max, floating-point rate, 0/1 setMissing, string invalidQuality)	TimeSeriesMath	7.15.60
setCurve(string name)	None	7.15.61
setCurve(integer number)	None	7.15.62
setData(DataContainer data)	None	7.15.63
setLocation(string location)	None	7.15.64
setParameter(string parameter)	None	7.15.65
setPathname(string pathname)	None	7.15.66
setTimeInterval(string interval)	None	7.15.67
setType(string type)	None	7.15.68
setUnits(string units)	None	7.15.69
setVersion(string version)	None	7.15.70
setWatershed(string watershed)	None	7.15.71
shiftAdjustment(TimeSeriesMath dataset)	TimeSeriesMath	7.15.72
shiftInTime(string timeShift)	TimeSeriesMath	7.15.73
sin()	HecMath	7.15.74
skewCoefficient()	floating-point	7.15.75
snapToRegularInterval(string timeInterval, string timeOffset, string timeBackward, string timeForward)	TimeSeriesMath	7.15.76
sqrt()	HecMath	7.15.77
standardDeviation()	floating-point	7.15.78
straddleStaggerRouting(integer avgCount, integer lagCount, integer subReachCount)	TimeSeriesMath	7.15.79
subtract(floating-point constant)	HecMath	7.15.80
subtract(TimeSeriesMath dataset)	TimeSeriesMath	7.15.81
successiveDifferences()	TimeSeriesMath	7.15.82
sum()	floating-point	7.15.83
tan()	HecMath	7.15.84
timeDerivative()	TimeSeriesMath	7.15.85
transformTimeSeries(string timeInterval, string timeOffset, string functionType)	TimeSeriesMath	7.15.86

Method	Returns	Description Section
transformTimeSeries(TimeSeriesMath dataset, string functionType)	TimeSeriesMath	7.15.87
truncate()	HecMath	7.15.88
twoVariableRatingTableInterpolation(TimeSeriesMath dataset1, TimeSeriesMath dataset2)	TimeSeriesMath	7.15.89

7.8 DataContainer Class

HecMath objects – data sets read from DSS files - cannot be added to plots and tables directly. Instead, DataContainer objects must be extracted from the HecMath objects for adding to plots or tables. The TimeSeriesContainer and PairedDataContainer classes discussed below are both types of DataContainer classes and are extracted from HecMath objects using the getData() method as documented in Section 7.15.24 (Get Data Container). HecMath objects can be updated from DataContainer objects using the setData() method as documented in Section 7.15.63 (Set Data Container). DataContainer objects have no methods that can be called by the user, but all data fields of DataContainer objects are directly accessible. Table 7.7 describes DataContainer data fields.

Table 7.7 – DataContainer Data Fields

Field	Type	Description
fullName	string	The full name associated with the data in the data store (DSS pathname,if the DataContainer is associated with a DSS file)
location	string	The location associated with the data (DSS pathname B-part if the DataContainer is associated with a DSS file).
subVersion	string	The sub-version associated with the data.
version	string	The version associated with the data (DSS pathname F-part if the DataContainer is associated with a DSS file).
watershed	string	The watershed associated with the data (DSS pathname A-part if the DataContainer is associated with a DSS file).

Example 18: Extracting and Using a DataContainer Object

```
from hec.script import *
from hec.hecmath import *
theFile = DSS.open(“myFile.dss”)
flowDataSet = theFile.read(“/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/")
theFile.close()
flowData = flowDataSet.getData()
thisWatershed = flowData.watershed
```

7.8.1 TimeSeriesContainer Class

TimeSeriesContainer is a type of DataContainer that contains information about time series data. TimeSeriesContainer objects are returned by the getData() method and are required in the setData() method of TimeSeriesMath objects. TimeSeriesContainer objects have all the data fields described Section 7.8 (DataContainerClass) in addition to those described in Table 7.8 . New TimeSeriesContainer objects can be created by a script if the TimeSeriesContainer class has been imported from the hec.io module (e.g. “from hec.io import *” or “from hec.io import TimeSeriesContainer”) The new object can be created by calling TimeSeriesContainer () (e.g. "myTSC = TimeSeriesContainer ()").

Table 7.8 – TimeSeriesContainer Data Fields

Field	Type	Description
endTime	integer[1]	The end time of the time window. If the data were retrieved, the end time may be later than the last time in the times list.
interval	Integer	The interval, in minutes, between each set of consecutive times in the times list. For irregular-interval times, the interval field is set to –1.
numberValues	Integer	The length of values and times lists.
parameter	string	The parameter associated with the data.
quality	list of integers[2]	The optional list of quality flags. If this list is present, there must be a quality for each value in the values array. If this list is not present, the quality field is set to None.
startTime	integer[1]	The start time of the time window. If the data were retrieved, the start time may be earlier than the first time in the times list.
subLocation	string	The sub-location associated with the data.
subParameter	string	The sub-parameter associated with the data.
times	list of integers[1]	The list of times. There must be a time for each value in the values list, and times must increase from one index to the next.
timeZoneID	string	The time-zone for times in the times list. If unknown, the timeZoneID field is set to None
timeZoneRawOffset	integer	The offset, in milliseconds, from UTC to the time zone for the times in the times list.
type	string	The type of the data (e.g. "INST-VAL", "INST-CUM", "PER-AVER", "PER-CUM").
units	String	The units of the data.
values	list of floating-point	The data values, each of which has a corresponding time in the times list and optionally a corresponding quality in the quality list. All lists must be the same length.

[1] Integer times from this field can be converted to string representations by using the set() and dateAndTime() methods of HecTime objects discussed in Section 7.9 (HecTime class). Additionally, integer times suitable for this field can be generated by the HecTime value() method.

[2] Quality values are interpreted according to **Table 7.9**

.**Table 7.9 – Data Quality Bits**

Bit(s)	Description
1	Set when the datum has been tested (screened).
2	Set when the datum passed all tests.
3	Set when the datum is missing (either originally missing or set to missing by a test).
4	Set when at least one test classified the datum as questionable.
5	Set when at least one test classified the datum as rejected.
6-7	Set by the RANGE test. Interpreted as a 2-bit unsigned integer with values having the following meanings: **0** value of datum < than 1st limit **1** 1st limit <= value of datum < 2nd limit **2** 2nd limit <= value of datum < 3rd limit **3** 3rd limit <= value of datum
8	Set when the datum has been changed from the original value.
9-11	Datum replacement indicator. Interpreted as a 3-bit unsigned integer with values having the following meanings: **0** datum was not replaced (original value) **1** datum was replaced by DATCHK **2** datum was replaced by DATVUE **3** datum was replaced by manual edit in DATVUE **4** original value was accepted in DATVUE **5-7** reserved for future use
12-15	Datum replacement value computation method. Interpreted as a 4-bit unsigned integer with values having the following meanings: **0** datum was not replaced (original value) **1** datum value computed by linear interpolation **2** datum value was entered manually **3** datum value was replaced with a missing value **4-15** reserved for future use
16	set when datum failed an absolute magnitude test
17	set when datum failed a constant value test
18	set when datum failed a rate of change test
19	set when datum failed a relative magnitude test
20	set when datum failed a duration magnitude test
21	set when datum failed a negative incremental value test
22	reserved for future use
23	set when datum is excluded from testing (e.g. DATCHK GAGEFILE entry)
24	reserved for future use
25	set when datum failed a user-defined test
26	set when datum failed a distribution test

Bit(s)	Description
27-31	reserved for future use
32	set when datum is protected from being replaced

Example 19: Using a TimeSeriesContainer Object

```
from hec.script import *
from hec.hecmath import *
from hec.io import TimeSeriesContainer
from hec.heclib.util import HecTime

watershed = "GREEN RIVER"
loc = "OAKVILLE"
param = "STAGE"
ver = "OBS"
startTime = "12Oct2003 0100"
values = [12.36, 12.37, 12.42, 12.55, 12.51, 12.47, 12.43, 12.39]
hecTime = HecTime()
tsc = TimeSeriesContainer()
tsc.watershed = watershed
tsc.location = loc
tsc.parameter = param
tsc.version = ver
tsc.fullname = "/%s/%s/%s//1HOUR/%s/" % \
    (watershed, loc, param, ver)
tsc.interval = 60
hecTime.set(startTime)
times = []
for value in values :
    times.append(hecTime.value())
    hecTime.add(tsc.interval)
tsc.values = values
tsc.times = times
tsc.startTime = times[0]
tsc.endTime = times[-1]
tsc.numberValues = len(values)
tsc.units = "FEET"
tsc.type = "INST-VAL"
stageRecord = HecMath.createInstance(tsc)
dssFile = DSS.open("myFile.dss")
dssFile.write(stageRecord)
dssFile.close()
```

7.8.2 PairedDataContainer Class

PairedDataContainer is a type of DataContainer that contains information about paired data. PairedDataContainer objects are returned by the getData() method and are required in the setData() method of PairedDataMath objects. TimeSeriesContainer objects have all the data fields described Section 7.8 (

DataContainer Class) in addition to those described in Table 7.10. New PairedDataContainer objects can be created by a script if the PairedDataContainer class has been imported from the hec.io module (e.g. "from hec.io import *" or "from hec.io import PairedDataContainer") The new object can be created by calling PairedDataContainer () (e.g. "myPDC = PairedDataContainer ()").

Table 7.10 – PairedContainer Data Fields

Field	Type	Description
date	string	The date associated with the paired-data.
datum	floating-point	The zero-stage elevation of a stream gauge.
labels	list of strings	The list of labels used to identify each of the y-ordinates lists. If there is only 1 y-ordinates list (e.g. numberCurves == 1), the labels field is set to None.
labelsUsed	0/1	A flag specifying whether labels are used. This field is set to Constants.TRUE if there is more than 1 y-ordinates list, and Constants.FALSE otherwise.
numberCurves	integer	The number of y-ordinates lists.
numberOrdinates	integer	The length of the x-ordinates list and each of the y-ordinates lists.
offset	floating-point	The offset value of a stream rating.
shift	floating-point	The shift value for a stream rating.
transformType	integer	Type of transformation to use (1 = "LINLIN", 2 = "LOGLOG")
xOrdinates	list of floating-point	The x-ordinate values. Each y-ordinate values list must be of the same length as this field.
xparameter	string	The parameter of the x-ordinates.
xtype	string	The type of the x-ordinates ("UNT" for unitary or "LOG" for logarithmic).
xunits	string	The units of the x-ordinates.
yOrdinates	list of lists of floating-point	The y-ordinate values. Each list of y-ordinate values must be of the same length as the list of x-ordinate values. The nth value of each y-ordinates list is associated with the nth value of the x-ordinates list.
yparameter	string	The parameter of the y-ordinates.
ytype	string	The type of the y-ordinates ("UNT" for unitary or "LOG" for logarithmic).
yunits	string	The units of the y-ordinates.

Example 20: Using a PairedDataContainer Object

```
from hec.script import *
from hec.hecmath import *
from hec.io import TimeSeriesContainer

watershed = "GREEN RIVER"
loc = "OAKVILLE"
xParam = "STAGE"
yParam = "FLOW"
date = "12Oct2003"
stages = [0.4, 0.5, 1.0, 2.0, 5.0, 10.0, 12.0]
flows = [0.1, 3, 11, 57, 235, 1150, 3700]
pdc = PairedDataContainer()
pdc.watershed = watershed
pdc.location = loc
pdc.parameter = param
pdc.version = ver
pdc.fullname = "/%s/%s/%s-%s///%s/" % \
    (watershed, loc, xParam, yParam, date)
pdc.xOrdinates = stages
pdc.yOrinates = [flows]
pdc.numberCurves = 1
pdc.numberOrdinates = len(stages)
pdc.labelsUsed = Constants.FALSE
pdc.xUnits = "FEET"
pdc.yUnits = "CFS"
pdc.xType = "LOG"
pdc.yType = "LOG"
pdc.xParameter = xParam
pdc.yParameter = yParam
pdc.date = date
pdc.transformType = 2

rating = HecMath.createInstance(pdc)
dssFile = DSS.open("myFile.dss")
dssFile.write(rating)
dssFile.close()
```

7.9 HecTime Class

HecTime objects are used to manipulate dates and times and to convert dates and times among different formats. To use HecTime objects, the HecTime class must be imported from the hec.heclib.util module (e.g "from hec.heclib.util import *" or "from hec.heclib.util import HecTime"). After importing the class, a new HecTime object can be created by calling HecTime() (e.g. "myTime = HecTime()").Table 7.11 describes **HecTime** methods.

Table 7.11: HecTime Methods

Method	Returns	Description
add(HecTime increment)	None	Adds the specified increment to the object's date and time.
add(integer increment)	None	Adds the specified increment in minutes to the object's date and time.
compareTimes(HecTime other)	integer	Returns one of the following values: **-1** The object's date and time is less than the other object's **0** The objects' dates and times are equal **1** The object's date and time is greater than the other object's
date()	string	Returns a string representation of the object's date. Same as date(2).
date(integer format)	string	Returns a string representation of the object's date, formatted according to the integer parameter[1].
dateAndTime()	string	Returns a string representation of the object's date and time. Same as dateAndTime(2).
dateAndTime(integer format)	string	Returns a string representation of the object's date and time, formatted according to the integer parameter[1].
day()	integer	Returns the day portion of the object's date as an integer
equalTo(HecTime other)	0/1	Returns Constants.TRUE if the object's date and time is equal to the other object's date and time. Returns Contstants.FALSE otherwise.

Method	Returns	Description
greaterThan(HecTime other)	0/1	Returns Contstants.TRUE if the object's date and time is greater (later) than the other object's date and time. Returns Constants.FALSE otherwise.
greaterThanEqualTo(HecTime other)	0/1	Returns Contstants.TRUE if the object's date and time is greater (later) than or equal to the other object's date and time. Returns Constants.FALSE otherwise.
hour()	integer	Returns the hours portion of the object's time as an integer
isDefined()	0/1	Returns Constants.TRUE if the object is set to a valid time and Constants.FALSE if not.
lessThan(HecTime other)	0/1	Returns Contstants.TRUE if the object's date and time is less (earlier) than the other object's date and time. Returns Constants.FALSE otherwise.
lessThanEqualTo(HecTime other)	0/1	Returns Contstants.TRUE if the object's date and time is less (earlier) than or equal to the other object's date and time. Returns Constants.FALSE otherwise.
minute()	integer	Returns the minutes portion of the object's time as an integer
month()	integer	Returns the month portion of the object's date as an integer
notEqualTo(HecTime other)	0/1	Returns Constants.FALSE if the object's date and time is equal to the other object's date and time. Returns Contstants.TRUE otherwise.
second()	integer	Returns the seconds portion of the object's time as an integer
set(integer time)	None	Sets the object to the date and time represented by the integer (minutes since 31Dec1899 00:00)
set(string dateAndTime)	integer	Sets the object to the date and time represented by the string, and returns zero if successful.
set(string date, string time)	integer	Sets the object to the date represented by the date string and the time represented by the time string, and returns zero if successful.

Method	Returns	Description
set(HecTime time)	None	Sets the object to the date and time represented by the HecTime parameter.
setDate(string date)	integer	Sets the object to the date represented by the string, and returns zero if successful. The time portion of the object is not modified.
setTime(string time)	integer	Sets the object to the time represented by the string, and returns zero if successful. The date portion of the object is not modified.
setUndefined()	None	Sets the object to represent an undefined time, as if the object had just been created.
showTimeAsBeginningOfDay(0/1 showBeginning)	None	Specifies whether the object is to show midnight times as 00:00 (vs 24:00)
subtract(HecTime increment)	None	Subtracts the specified increment from the object's date and time.
subtract(integer increment)	None	Subtracts the specified increment in minutes from the object's date and time.
time()	string	Returns a string representation of the object's time.
value()	integer	Returns the object's date and time as an the number of minutes since 31Dec1899 0000.
year()	integer	Returns the year portion of the object's date as an integer

[1] The format of the string returned by the date(integer format) method and the date portion of the string returned by the dateAndTime(integer format) method are displayed in Table 7.12.

Table 7.12: HecTime Date Formats

0	`June 2, 1985`	**10**	`June 2, 85`	**100**	`JUNE 2, 1985`	**110**	`JUNE 2, 85`
1	`Jun 2, 1985`	**11**	`Jun 2, 85`	**101**	`JUN 2, 1985`	**111**	`JUN 2, 85`
2	`2 June 1985`	**12**	`2 June 85`	**102**	`2 JUNE 1985`	**112**	`2 JUNE 85`
3	`June 1985`	**13**	`June 85`	**103**	`JUNE 1985`	**113**	`JUNE 85`
4	`02Jun1985`	**14**	`02Jun85`	**104**	`02JUN1985`	**114**	`02JUN85`
5	`2Jun1985`	**15**	`2Jun85`	**105**	`2JUN1985`	**115**	`2JUN85`
6	`Jun1985`	**16**	`Jun85`	**106**	`JUN1985`	**116**	`JUN85`
7	`02 Jun 1985`	**17**	`02 Jun 85`	**107**	`02 JUN 1985`	**117**	`02 JUN 85`
8	`2 Jun 1985`	**18**	`2 Jun 85`	**108**	`2 JUN 1985`	**118**	`2 JUN 85`
9	`Jun 1985`	**19**	`Jun 85`	**109**	`JUN 1985`	**119**	`JUN 85`

7.10 Plotting Basics

Figure 7.6 identifies the title, viewport, axis label, axis tics, and legend of a plot, each of which are accessible via scripts.

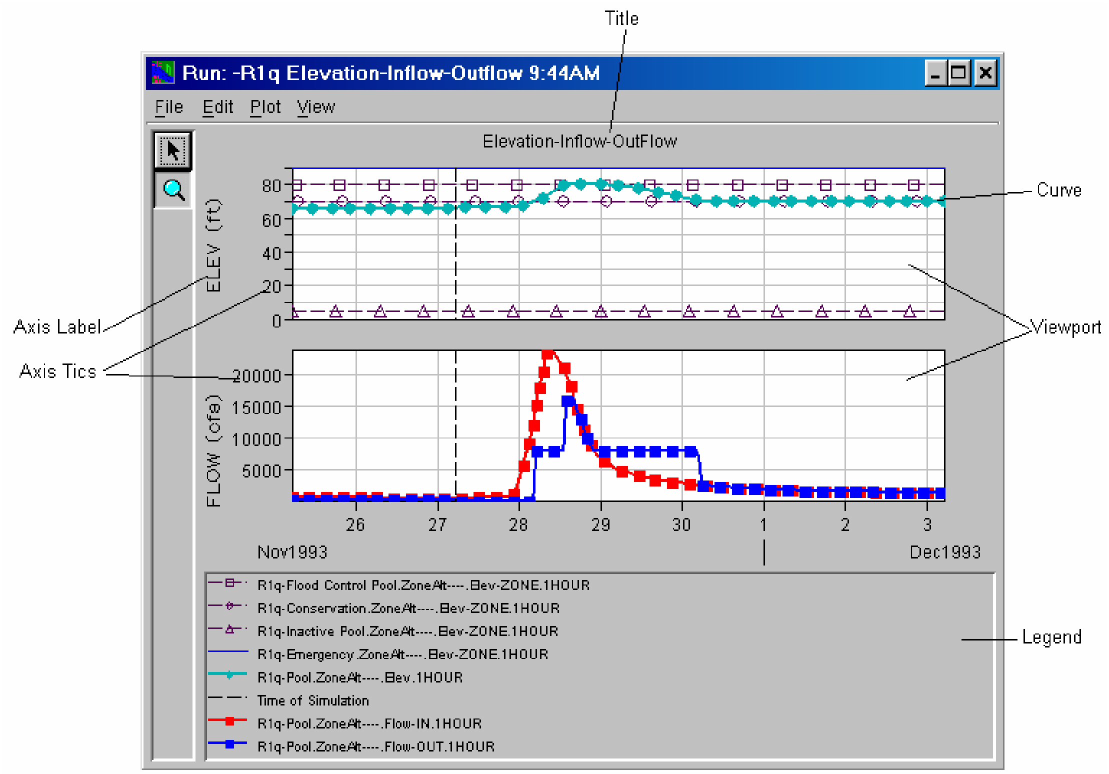

Title
Run: -R1q Elevation-Inflow-Outflow 9:44AM
File Edit Plot View
Elevation-Inflow-OutFlow
Curve
ELEV (ft)
80
60
40
20
0
Viewport
Axis Label
Axis Tics
FLOW (cfs)
20000
15000
10000
5000
26
27
28
29
30
1
2
3
Nov1993
Dec1993
R1q-Flood Control Pool.ZoneAlt----.Elev-ZONE.1HOUR
R1q-Conservation.ZoneAlt----.Elev-ZONE.1HOUR
R1q-Inactive Pool.ZoneAlt----.Elev-ZONE.1HOUR
R1q-Emergency.ZoneAlt----.Elev-ZONE.1HOUR
R1q-Pool.ZoneAlt----.Elev.1HOUR
Time of Simulation
R1q-Pool.ZoneAlt----.Flow-IN.1HOUR
R1q-Pool.ZoneAlt----.Flow-OUT.1HOUR
Legend

Figure 7.6 Plot Components

7.10.1 Plot Class

```
Plot.newPlot()
Plot.newPlot(string title)
```

The Plot class in the hec.script module is used to create a new Plot dialog. It contains two methods to create a Plot dialog, each of which returns a G2dDialog object.

Example 21: Creating a Plot

```
myPlot = Plot.newPlot()
```

or

```
thePlot = Plot.newPlot("Elevation vs Flow")
```

7.10.2 Changing Plot Component Attributes

Not all Plot Component attributes are visible by default, and setting the attribute may not make that attribute visible. Often it is necessary to set the visibility of the attribute by calling set*Attribute*Visible(Constants.TRUE) method.

Example 22 illustrates reading a flow data set from a DSS file, plotting the data set, setting the minor Y grid color to black and making it display.

Example 22: Plotting DSS Data

```
from hec.script import *
from hec.script.Constants import TRUE, FALSE
from hec.hecmath import *

theFile = DSS.open("myFile.dss")              # open myFile.dss
thePath = "/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowDataSet = theFile.read(thePath)           # read a path name
thePlot = Plot.newPlot()                      # create the plot
thePlot.addData(flowDataSet.getData())        # add the flow data set to
                                              the plot
thePlot.showPlot()                            # show the plot
viewport0=thePlot.getViewport(0)              # get the first viewport
viewport0.setMinorGridYColor("black")         # set the viewport's minor Y
                                              grid to black
viewport0.setMinorGridYVisible(TRUE)          # tell the minor Y grid to
                                              display
```

7.10.3 G2dDialog Class

G2dDialog objects are the dialog that plots display in. Table 7.13 describes **G2dDialog** methods.

Table 7.13: G2dDialog Methods

Method	Returns	Description
addData(DataContainer dc)	None	Add the DataContainer specified by dc to the plot. Must be called before showPlot()
applyTemplate(string templateFile)	None	Apply the given template to this plot
configurePlotLayout()	None	Display the "Configure Plot Layout" dialog for this plot
configurePlotLayout(PlotLayout layout)	None	Configures the plot layout for this plot according to the specifed PlotLayout object
close()	None	Closes the plot
configurePlotTypes()	None	Display the configure plot types dialog
copyToClipboard()	None	Copy the plot to the system clipboard
defaultPlotProperties()	None	Display the default plot properties dialog
exportProperties()	None	Allows you to save the properties of the plot to a disk.
exportProperties(string templateName)	None	Allows you to save the properties of the plot to the file specified by templateName.
getCurve(HecMath dataSet)	G2dLine	Return the G2dLine for the DataSet specified by dataSet
getCurve(string dssPath)	G2dLine	Return the G2dLine for the path specified in dssPath
getHeight()	integer	Return the height of the dialog in screen coordinates.
getLegendLabel(DataContainer dc)	G2dLabel	Return the legend label object for the specified data container.

Method	Returns	Description
getLocation()	Point	Return the location of the dialog in screen coordinates[1].
getPlotTitle()	G2dTitle	Return the title for the G2dDialog
getPlotTitleText()	string	Return the text of the title for the G2dDialog
getSize()	Dimension	Return the dimensions of the dialog in screen coordinates.
getViewport(HecMath dataSet)	Viewport	Return the Viewport that contains the curve specified by dataSet
getViewport(int viewportIndex)	Viewport	Return the viewport at index specified by viewportIndex
getViewport(string dataSetPath)	Viewport	Return the Viewport that contains the curve specified by dataSetPath
getWidth()	Integer	Return the width of the dialog in screen coordinates.
hide()	None	Hide the dialog
iconify()	None	Minimize (iconify) the dialog
isPlotTitleVisible()	0/1	Return the visibility state of the title of this plot.
maximize()	None	Maximize the dialog
minimize()	None	Minimize (iconify) the dialog
newPlotLayout()	PlotLayout	Return a PlotLayout object that can be used to configure the layout of this plot.
plotProperties()	None	Display the plot properties dialog for this plot
print()	None	Display the print dialog for this plot
printMultiple()	None	Display the print multiple dialog for this plot
printPreview()	None	Display the print preview dialog for this plot

Method	Returns	Description
printToDefault()	None	Prints using the printer defaults such as page format and printer. This method does not display the printer dialog for user interaction.
repaint()	None	Forces the plot to be refreshed.
restore()	None	Restore the dialog from a minimized or maximized state
saveAs()	None	Display the saveAs dialog for this plot
saveToJpeg(string fileName)	None	Save the plot to the Jpeg file specified by fileName
saveToJpeg(string fileName, integer quality)	None	Save the plot to the Jpeg file specified by filename, with the specified quality[2].
saveToMetafile(string filename)	None	Save the plot to the Windows Meta file specified by filename
saveToPng(string fileName)	None	Save the plot to the Portable Network Graphics file specified by filename
saveToPostscript(string fileName)	None	Save the plot to the PostScript file specified by filename
setLegendBackgound(string color)	None	Sets the background (fill) color of the legend.
setLegendLabelText(DataContainer dc, string text)	None	Sets the legend label text for the specified data container.
setLegendLocation(string location)	None	Sets the location of the legend[3].
setLocation(integer x,integer y)	None	Sets the location of the dialog in screen coordinates[1].
setPlotTitleText(string text)	None	Sets the text of the title for this plot
setPlotTitleVisible(0/1 state)	None	Sets the visibility of the title for this plot
setSize(integer width, integer height)	None	Sets the size of the dialog in screen coordinates.

Method	Returns	Description
setVisible(0/1 visible)	None	Sets the visibility of the plot to true or false
showPlot()	None	Show the dialog
tabulate()	TableFrame	Display the table view of this plot

[1] The coordinate system used is a graphics coordinate system, where X increases to the right and Y increases downward from the origin (0,0) which is located in the top left corner of the display. Locations set or retrieved refer to the top left corner of the plot in reference to this coordinate system.

[2] The specified quality is limited to an effective range of 0 – 100, inclusive. Higher qualities produce larger files and take longer to generate. The saveToJpeg(fileName) call currently produces the same results as saveToJpeg(fileName, 75).

[3] Valid legend locations are “Right” and “Bottom”.

Example 23 shows how to create a new plot with a flow data set, show the plot, and place at location 50,50 on the screen.

Example 23: Plot Dialog

```
from hec.script import *                      # for Plot class
from hec.hecmath import *                     # for DSS class
theFile = DSS.open(“myFile.dss”)              # open myFile.dss
thePath = “/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowDataSet = theFile.read(thePath)           # read a path name

dc = flowDataSet.getData()                    # create flow data container
thePlot = Plot.newPlot()                      # create a new Plot
thePlot.addData(dc)                           # add flow data container
thePlot.showPlot()                            # show the plot
thePlot.setLocation(50,50)                    # moves plot to 50,50
```

7.10.4 PlotLayout Class

PlotLayout objects hold information about the layout of the plot dialog. The use of ViewportLayout objects, in conjuction with PlotLayout objects, allows scripts to specify the same layout information accessible interactive via the "Configure Plot Layout" dialog. A PlotLayout object is obtained by calling Plot.newPlotLayout(). Table 7.14 describes **PlotLayout** methods.

Table 7.14: PlotLayout Methods

Method	Returns	Description
addViewport()	ViewportLayout	Adds a ViewportLayout to the PlotLayout with a default weight of 100. Returns a reference to the new ViewportLayout.
AddViewport(floating-point weight)	ViewportLayout	Adds a ViewportLayout to the PlotLayout with the specified weight. Returns a reference to the new ViewportLayout.
hasLegend()	0/1	Returns whether this PlotLayout is configured to display the legend.
hasToolbar()	0/1	Returns whether this PlotLayout is configured to display the toolbar.
getViewportCount()	integer	Returns the number of ViewportLayout objects currently in the PlotLayout object.
getViewports()	java.util.List of ViewportLayouts	Returns the ViewportLayout objects currently in the PlotLayout object.
getViewportWeights()	list of floating-points	Returns the weights of the ViewportLayout objects currently in the PlotLayout object.
setHasLegend(0/1 state)	None	Configures the PlotLayout object to display the legend or not, depending upon the specified state.
setHasToolbar(0/1 state)	None	Configures the PlotLayout object to display the toolbar or not, depending upon the specified state.

7.10.5 ViewportLayout Class

ViewportLayout objects hold information about the layout of an individual viewport withing the plot dialog. The use of ViewportLayout objects, in conjuction with PlotLayout objects, allows scripts to specify the same layout information accessible interactive via the "Configure Plot Layout" dialog. A ViewportLayout object is obtained by calling one of the addViewport methods of a PlotLayout object. ViewportLayout objects are only used to configure the plot layout. Manipulation of axis labels, background colors, etc… is performed using Viewport objects as described in below. Table 7.15 describes **ViewportLayout** methods.

Table 7.15: ViewportLayout Methods

Method	Returns	Description
addCurve(string axis, DataContainer curve)	None	Adds the specified curve to the specified axis of the ViewportLayout object.
getY1Data()	List	Returns a java.util.List of all curves that have been added to the Y1 axis of this object
getY2Data()	List	Returns a java.util.List of all curves that have been added to the Y2 axis of this object
hasY1Data()	0/1	Returns whether any curves have been added to the Y1 asix of this object
hasY2Data()	0/1	Returns whether any curves have been added to the Y2 asix of this object

Example 24 reads precipitation, stage and flow data set from a DSS file, and configures a plot to display the precipitation on top in a viewport that occupies 30% of the available space and to display the stage and flow on separate axes of a bottom viewport that occupies the remaining 70% of available space.

Example 24: PlotLayout and ViewportLayout Objects

```
from hec.script import *                          # for Plot class
from hec.hecmath import *                         # for DSS class
theFile = DSS.open(“myFile.dss”)                  # open myFile.dss
precipPath = “/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
stagePath = “/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowPath = “/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
precipData = theFile.read(precipPath)             # read the precip
stageData = theFile.read(stagePath)               # read the stage
flowData = theFile.read(flowPath)                 # read the flow
precipDC = precipData.getData()                   # extract data container
stageDC = stageData.getData()                     # extract data container
flowDC = flowData.getData()                       # extract data container
thePlot = Plot.newPlot()                          # create a new Plot
layout = Plot.newPlotLayout()                     # create a new PlotLayout
topView = layout.addViewport(30)                  # get the top viewport
bottomView = layout.addViewport(70)               # get the bottom viewport
topView.addCurve(“Y1”, precipDC)                  # add the precip to top
bottomView.addCurve(“Y1”, stageDC)                # add the stage to bottom
bottomView.addCurve(“Y2”, flowDC)                 # add the flow to bottom
thePlot.configurePlotLayout(layout)               # configure the plot
thePlot.showPlot()                                # show the plot
```

7.10.6 Viewport Class

Viewport objects hold the data set curves. Table 7.16 describes **Viewport** methods.

Table 7.16: Viewport Methods

Method	Returns	Description
addAxisMarker(AxisMarker marker)	None	Adds a marker line described by the AxisMarker parameter
addXAxisMarker()	None	Display the Axis Marker Properties Dialog for a marker line to add to the X axis
addXAxisMarker(floating-point value)	None	Add an X Axis marker at the location specified by value
addXAxisMarker(string value)	None	Add a X Axis marker at the location specified by value

Method	Returns	Description
addYAxisMarker()	None	Display the Axis Marker Properties Dialog for a marker line to add to the Y axis
addYAxisMarker(string value)	None	Add a Y Axis marker at the location specified by value
editProperties()	None	Display the Edit Properties dialog for this Viewport
getAxis(string axisName)	Axis	return the Axis specified by axisName for this Viewport
getAxisLabel(string axisName)	AxisLabel	Return the AxisLabel for the axis specified by axisName for this Viewport
getAxisTics(string axisName)	AxisTics	Return the AxisTics for the axis specified by axisName for this Viewport
getBackground()	Color	Return the background color for the Viewport as a Color.
getBackgroundString()	string	Return the background color name for the Viewport as a string.
getBorderColor()	Color	Return the border color for the Viewport as a Color.
getBorderColorString()	string	Return the background color name for the Viewport as a string
getBorderWeight()	float	Return the border weight for this Viewport
getFillPatternString()	string	Return the fill pattern for this Viewport as a String
getMajorGridXColor()	Color	Return the color of the vertical lines of the major grid for this Viewport as a Color
getMajorGridXColorString ()	string	Return the color of the vertical lines of the major grid for this Viewport as a string

Method	Returns	Description
getMajorGridXWidth ()	floating point	Return the width of the vertical lines of the major grid for this Viewport
getMajorGridYColor ()	Color	Return the color of the horizontal lines of the major grid of this Viiewport as a Color
getMajorGridYColorString ()	string	Return the color of the horizontal lines of the major grid for this Viewport as a string
getMajorGridYWidth ()	floating point	Return the width of the vertical lines of the major grid for this Viewport
getMinorGridXColor()	Color	Return the color of the vertical lines of the minor grid for this Viewport as a color
getMinorGridXColorString()	string	Return the color of the vertical lines of the minor grid for this Viewport as a string
getMinorGridXWidth()	floating point	Return the width of the vertical lines of the minor grid for this Viewport
getMinorGridYColor()	Color	Return the color of the horizontal lines of the minor grid for this Viewport as a Color.
getMinorGridYColorString()	string	Return the color of the horizontal lines of the minor grid for this Viewport as a string
getMinorGridYWidth()	floating point	Return the width of the vertical lines of the minor grid for this Viewport
isBackgroundVisible()	0/1	Return whether the background is drawn for this Viewport
isBorderVisible()	0/1	Return whether the border is drawn for this Viewport
isMajorGridXVisible()	0/1	Return whether the vertical lines of the major grid are drawn for this Viewport

Method	Returns	Description
isMajorGridYVisible ()	0/1	Return whether the horizontal lines of the major grid are drawn for this Viewport
isMinorGridXVisible()	0/1	Return whether the vertical lines of the minor grid are drawn for this Viewport
isMinorGridYVisible ()	0/1	Return whether the horizontal lines of the minor grid are drawn for this Viewport
setBackground(string colorString)	None	Set the background to the color specified by colorString
setBorderColor(string borderColor)	None	Set the border color for this Viewport
setBorderWeight(floating-point borderWeight)	None	Set the border weight for this Viewport
setBackgroundVisible(0/1 state)	None	Set whether to draw the background for this Viewport
setBorderVisible(0/1 state)	None	Set whether to draw the border for this Viewport
setFillPattern(string pattern)	None	Set the fill pattern for this Viewport
setGridColor(string colorString)	None	Set the color of the horizontal and vertical lines of the major and minor grids for this Viewport.
setGridXColor(string colorString)	None	Set the color of the vertical lines of the major and minor grids for this Viewport.
setGridYColor(string colorString)	None	Set the color of the horizontal lines of the major and minor grids for this Viewport.
setMajorGridXColor(string majorGridXColor)	None	Set the color of the vertical lines of the major grid for this Viewport.
setMajorGridXVisible(0/1 state)	None	Set whether to draw the vertical lines of the major grid for this Viewport

Method	Returns	Description
setMajorGridXWidth(floating-point gridLineWidth)	None	Set the width of the vertical lines of the major grid for this Viewport
setMajorGridYColor(string majorGridYColor)	None	Set the color of the horizontal lines of the major grid for this Viewport.
setMajorGridYVisible(0/1 state)	None	Set whether to draw the horizontal lines of the major grid for this Viewport
setMajorGridYWidth(floating-point gridLineWidth)	None	Set the width of the horizontal lines of the major grid for this Viewport
setMinorGridXColor(string minorGridXColor)	None	Set the color of the vertical lines of the minor grid for this Viewport.
setMinorGridXVisible(0/1 state)	None	Set whether to draw the vertical lines of the minor grid for this Viewport
setMinorGridXWidth(floating-point gridLineWidth)	None	Set the width of the vertical lines of the minor grid for this Viewport
setMinorGridYColor(string minorGridYColor)	None	Set the color of the horizontal lines of the minor grid for this Viewport.
setMinorGridYVisible(0/1 state)	None	Set whether to draw the horizontal lines of the minor grid for this Viewport
setMinorYGridWidth(floating-point gridLineWidth)	None	Set the width of the horizontal lines of the minor grid for this Viewport

7.10.7 AxisMarker Class

AxisMarker objects hold complete descriptions of marker lines to be added to viewports. AxisMarker objects have fields that are settable by the user to create marker lines of various styles. New AxisMarker objects are created by calls to `AxisMarker()` (e.g. `myMarker = AxisMarker()`).

Table 7.17 describes **AxisMarker** fields.

Table 7.17 – AxisMarker Fields

Field	Type	Description	Default
axis	string	"X" or "Y"	"Y"
fillColor	string	Color of the filled area	"black"
fillPattern	string	Pattern of the filled area	"solid"
fillStyle	string	Specifies whether the filled area is to be above or below the marker line, or to not fill at all	"none"
labelAlignment	string	Specifies whether the label text is to appear left justified, right justified or centered	"left"
labelColor	string	Color of the label text	"black"
labelFont	string	The font to use for the label[1].	None
labelPosition	string	Specifies whether the label text is to appear above, below, or in the center of the marker line	"above"
labelText	string	Text to appear with marker line	""
lineColor	string	Color of the marker line	"black"
lineStyle	string	Style of the marker line	"solid"
lineWidth	floating point	Width of the marker line	1.0
value	string	Location of marker on axis (e.g. "712.5" or "23Aug2003 1015")	"0"

[1] Fonts are specified as name[,style[,size]] where style is Plain, Bold, Italic, or BoldItalic (e.g. "Arial,BoldItalic,12", "Lucida Console,Plain,10").

Example 25 reads a data set from a DSS file, plots that data set, sets the Viewport's background to light gray and adds a marker line on the Y axis.

Example 25: Viewport Class

```
from hec.script import *                            # for Plot class
from hec.script.Constants import TRUE, FALSE
from hec.hecmath import *                           # for DSS class
theFile = DSS.open(“myFile.dss”)                    # open myFile.dss
thePath = “/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowDataSet = theFile.read(thePath)                 # read a path name
thePlot = Plot.newPlot()                            # create a new Plot
thePlot.addData(flowDataSet.getData())              # add the flow data
viewport0=thePlot.getViewport(0)                    # get the first Viewport
viewport0.setBackground(“lightgray”)                # set the Viewport’s
                                                    # background to lightgray
viewport0.setBackgroundVisible(TRUE)                # tell the Viewport to draw
                                                    # its background
marker = AxisMarker()                               # create a new marker
marker.axis = “Y”                                   # set the axis
marker.value = “20000”                              # set the value
marker.labelText = “Damaging Flow”                  # set the text
marker.labelColor = “red”                           # set the text color
marker.lineColor = “red”                            # set the line color
marker.fillColor = “red”                            # set the fill color
marker.fillType = “above”                           # set the fill type
maker.fillPattern = “diagonal cross”                # set the fill pattern
viewport0.addAxisMarker(marker)                     # add the marker to the
                                                    # viewport
```

7.10.8 Axis Class

Table 7.18 describes **Axis** methods.

Table 7.18 - Axis Methods

Method	Returns	Description
getLabel()	string	Return the Axis label
getMajorTic()	floating-point	Return the major tic interval for this Axis
getMinorTic()	floating-point	Return the minor tic interval for this Axis
getNumTicLabelLevels()	integer	Return the number of tic label levels for this Axis
getScaledLabel()	String	Return the label with scientific notation
getScaleMax()	floating-point	Return the maximum value for this Axis
getScaleMin()	floating-point	Return the minimum value for this Axis
getTicColor()	Color	Return the tic color
getTicColorString()	String	Return the Tic color as a String
getTicTextColor()	Color	Return the tic text color
getTicTextColorString()	String	Return the tic text color as a String
getViewMax()	floating-point	Return the maximum value for the (possibly) zoomed view for this Axis
getViewMin()	floating-point	Return the minimum value for the (possibly) zoomed view for this Axis
isComputingMajorTics()	0/1	Return if major tics are to be computed
isComputingMinorTics()	0/1	Return if minor tics are to be computed
isReversed()	0/1	Returns whether the Axis is reversed[1].
setComputeMajorTics(0/1 state)	None	Set whether to compute major tics
setComputeMinorTics(0/1 state)	None	Set whether to compute minor tics
setLabel(string label)	None	Set the label of this Axis

Method	Returns	Description
setMajorTicInterval(floating-point interval)	None	Set the major tic interval for this Axis to interval
setMinorTicInterval(floating-point interval)	None	Set the minor tic interval for this Axis to interval
setNumTicLabelLevels(integer layers)	None	Set the maximum number of tic label layers to specified number. -1 is unrestricted. Used mostly for time series axis.
setReversed(0/1 state)	None	Set the reversed state of the Axis[1].
setScaleLimits(floating-point min, floating-point max)	None	Sets the minimum and maximum values for the axis (range of un-zoomed view)
setTicColor(String colorString)	None	Set the tic color to the color represented by colorString
setTicTextColor(String colorString)	None	Set the tic text color to the color represented by colorString
setViewLimits(floating-point min, floating-point max)	None	Zooms based on world coordinates
unZoom()	None	Returns the view to the full axis range.
zoomByFactor(floating-point factor)	None	Change the zoom scaling by the given factor

[1] The coordinate system used is a graphics coordinate system with the origin (0,0) located at the top left corner of the display, with X increasing to the right and Y increasing downward. The reversed state is in respect to this coordinate system (i.e. Y is reversed if it increases upward).

Example 26 reads a data set from a DSS file, adds that data set to a new Plot, and zooms in on the Y Axis.

Example 26: Using Axis Objects

```
from hec.script import *                          # for Plot class
from hec.hecmath import *                         # for DSS class
thePlot = Plot.newPlot()                          # create a Plot
dssFile = DSS.open(“J:/apps/forecast.dss”)        # open the DSS file
flow = dssFile.read(“/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/”)
                                                  # read a data set
thePlot.addData(flow.getData()                    # add the data set
thePlot.showPlot()                                # show the plot
viewport0 = thePlot.getViewport(0)                # get the first Viewport
yaxis = viewport0.getAxis(“Y1”)                   # get the Y1 axis
yaxis.setScaleLimits(0., 25000.)                  # set the scale
yaxis.zoomByFactor(.”5”)                          # zoom in
```

7.10.9 AxisTics Class

Table 7.19 describes **AxisTics** methods.

Table 7.19 – AxisTics Methods

Method	Returns	Description
areMajorTicLabelsVisible()	0/1	Return whether the major tic labels are visible.
areMajorTicsVisible()	0/1	Return whether the major tics are visible
areMinorTicLabelsVisible()	0/1	Return whether the minor tic labels are visible
areMinorTicsVisible()	0/1	Return whether the minor tics are visible
computeRatingFromOppositeAxis()	None	When used on the right (Y2) AxisTics object, with related curves on the Y1 and Y2 axes (e.g. stage and flow, or elevation and storage), causes the AxisTics to behave in a non-linear fashion such that Y1 and Y2 curves are coincident.
editProperties()	None	Display the Edit Properties Dialog for the AxisTics
getAxis()	Axis	Returns a reference to the axis that this object draws
getAxisTicColor()	Color	Return the tic color
getAxisTicColorString()	String	Return the tic color as a String

Method	Returns	Description
getFontSizes()	tuple of 3 integers	Return the regular, tiny, min and max font sizes for this AxisTics
getMajorTicLength()	Integer	Return the major tic length
getMinorTicLength()	integer	Return the minor tic length
setAxisTicColor(string colorString)	None	Set the tic color to the color represented by colorString
setFontSizes(integer sz,integer tiny, integer min, integer max)	None	Set the regular, tiny, min and max font sizes for this AxisTics
setMajorTicLabelsVisible(0/1 state)	None	Set the visibility of the major tic labels
setMajorTicLength(int ticLength)	None	Set the major tic length
setMajorTicsVisible(0/1 state)	None	Set the visibility of the major tics
setMinorTicLabelsVisible(0/1 state)	None	Set the visibility of the minor tic labels.
setMinorTicLength(int ticLength)	None	Set the minor tic length
setMinorTicsVisible(0/1 state)	None	Set the visibility of the minor tics

Example 27 creates a new Plot with a data set read from DSS and tells the data set’s axis tics to draw its minor tic marks.

Example 27: Using AxisTics Objects

```
from hec.script import *                      # for Plot class
from hec.script.Constants import TRUE, FALSE

from hec.hecmath import *                     # for DSS class
thePlot = Plot.newPlot()                      # create the Plot
dssFile = DSS.open("J:/apps/forecast.dss")    # open the DSS file
flow = dssFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/")
                                              # read the data set
thePlot.addData(flow.getData())               # add the data set
thePlot.showPlot()                            # show the plot
viewport0 = thePlot.getViewport(flow)         # get the viewport for the
                                              #flow data set
yAxisTics = viewport0.getAxisTics("Y1")       # get the axis tics for the
                                              #Viewport
yAxisTics.setMinorTicsVisible(TRUE)           # tell axis tics to show tics
```

7.10.10 G2dLine Class

Table 7.20 describes **G2dLine** methods

Table 7.20 - G2dLine Methods

Method	Returns	Description
areSymbolsAutoInterval()	0/1	Return whether the symbols for this line are placed at program-decided intervals
areSymbolsVisible()	0/1	Return whether this line draws its symbols
editLineProperties()	None	Method that allows the editing of line properties. This method displays a visible dialog for line editing.
getFillColor()	Color	Return the fill color for this line
getFillColorString()	string	Return the fill color for this line as a String
getFillPatternString()	string	Return the fill pattern for this line as a String
getFillTypeString()	string	Return the Fill type for this line as a String.
getFirstSymbolOffset()	integer	Return the offset for the first symbolfor this line
getLineColor()	Color	Return the line color for this line
getLineColorString()	string	Return the line color for this line as a String
getLineStepStyleString()	string	Return the line step style for this line as a String
getLineStyleString()	string	Return the line style for this line as a string
getLineWidth()	floating-point	Return the Line Width of the line
getNumPoints()	integer	Returns the Number of Points that this line has
getSymbolFillColor()	Color	Return the symbol fill color for this line's symbols
getSymbolFillColorString()	string	Return the symbol fill color for this line's symbols as a String
getSymbolInterval()	integer	Return the interval of data points (>0) on which symbols are drawn.

Method	Returns	Description
getSymbolLineColor()	Color	Return the symbol line color for this line's symbols
getSymbolLineColorString()	string	Return the symbol line color for this line's symbols as a String
getSymbolSize()	floating-point	Return the symbol size for this line
getSymbolSkipCountl()	integer	Return the number of points skipped between symbols (same as getSymbolInterval() – 1)
getSymbolTypeString()	string	Return the symbol type for this line as a string
isLineVisible()	0/1	Return this line is drawn
setFillColor(string fillColor)	None	Set the fill color for this line
setFillPattern(string fillPattern)	None	Set the fill pattern for this line
setFillType(string fillType)	None	Set the Fill type for this line
setFirstSymbolOffset(integer offset)	None	Set the offset for first symbol for this line
setLineColor(string lineColor)	None	Set the line color for this line
setLineStepStyle(string stepStyle)	None	Set the line step style for this line
setLineStyle(string style)	None	Set the line style for this line
setLineVisible(0/1 state)	None	Set whether to draw this line
setLineWidth(floating-point width)	None	Set the width for this line
setSymbolFillColor(string symbolFillColor)	None	Set the symbol fill color for this line's symbols
setSymbolInterval(integer interval)	None	Set the interval of data points (>0) on which symbols are drawn.
setSymbolLineColor(string symbolLineColor)	None	Set the symbol line color for this line's symbols
setSymbolsAutoInterval(0/1 state)	None	Set whether to have the program decide the interval at which to draw symbols
setSymbolSize(floating-point size)	None	Set the symbol size for this line
setSymbolSkipCount(integer count)	None	Set the number of points skipped between symbols.
setSymbolsVisible(0/1 state)	None	Set whether to draw the symbols for this line
setSymbolType(string symbolType)	None	Set the symbol type for this line

Example 28 creates a plot with a data set read from DSS, then tells that data set's curve to draw its symbols auto skipped.

Example 28: Using G2dLine Objects

```
from hec.script import *                    # for Plot class
from hec.hecmath import *                   # for DSS class
from hec.script.Constants import TRUE, FALSE

thePlot = Plot.newPlot()                    # create the Plot
file = "j:/apps/forecast.dss";
dssfile = DSS.open(file)                    # open the file
stage = dssfile.read("/BASIN/LOC/STAGE/01NOV2002/1HOUR/OBS/");
                                            # read the data set
thePlot.addData(stage.getData())            # add the data set to the
                                            plot

thePlot.showPlot()                          # show the plot
stageCurve = thePlot.getCurve(stage)        # get the stage curve
stageCurve.setSymbolsAutoInterval(TRUE)     # turn on symbols auto skip
```

7.10.11 G2dLabel, G2dTitle, and AxisLabel Classes

Table 7.21 describes **G2dLabel, G2dTitle and AxisLabel** methods.

Table 7.21 - Label Methods

Method	Returns	Description
editProperties()	None	Display the Edit Properties Dialog for the label
getAlignmentString()	string	Return the text alignment for this label as a String
getBackground()	Color	Return the background color for the label[1]
getBackgroundString()	string	Return the background color for the label as a String[1]
getBorderStyleString()	string	Return the border style for this label as a string
getBorderWeight()	floating-point	Return the border weight for this label
getFillColor()	Color	Return the fill color for this label as a Color[1]
getFillColorString()	string	Return the fill color for this label as a string[1]

Method	Returns	Description
getFillPatternString()	string	Return the fill pattern for this label as a string[1]
getFontFamily()	string	Return the font family for the label
getFontSize()	integer	Return the font size for the label
getFontSizes()	tuple of 3 integers	Return the regular, tiny, min and max font sizes for this label
getFontString()	string	Return the font for the label as a string[2].
getFontStyleString()	string	Return the font style for the label as a String
getForeground()	Color	Return the foreground color for the label
getForegroundString()	string	Return the foreground color for the label as a String
getIcon()	Icon	Return the Icon to display for this label
getIconPath()	string	Return the Icon path to display for this label
getRotation()	integer	Return the text rotation for this label
getSpacing()	integer	Return the spacing around this label
getText()	string	Return the text for the label
isBackgroundVisible()	0/1	Return whether the background is visible
isBorderVisible()	0/1	Return whether the border is visible
setAlignment(string alignment)	None	Set the text alignment for this label
setBackground(string colorString)	None	Set the background color for the label[1]
setBackgroundVisible(0/1 state)	None	Set the background visibility for the label
setBorderColor(string colorString)	None	Set the border color for this label
setBorderStyle(string style)	None	Set the border style for this label
setBorderVisible(0/1 state)	None	Set the border visibility for this label
setBorderWeight(floating-point weight)	None	Set the border weight for this label
setFillColor(string color)	None	Set the fill color for this label[1]
setFillPattern(string pattern)	None	Set the fill pattern for this label[1]
setFont(string font)	None	Set the font for the label[2].
setFontFamily(string fam)	None	Set the font family for the label
setFontSize(integer sz)	None	Set the font size for the label

Method	Returns	Description
setFontSizes(integer sz, integer tiny, integer min, integer max)	None	Set the regular, tiny, min and max font sizes for this label
setFontStyle(string style)	None	Set the font style for the label
setForeground(string colorString)	None	Set the foreground color for the label
SetIcon(Icon icon)	None	Set the Icon to display for this label
SetIcon(string iconPath)	None	Set the Icon to display for this label
setRotation(integer rotation)	None	Set the text rotation for this label
setSpacing(integer space)	None	Set the spacing around this label
SetText(string text)	None	Set the text for the label

[1] In the current version, fill color and background color are synonymous (e.g. fills are performed with the background color). Future version may support separate fill and background colors.

[2] Fonts are specified as name[,style[,size]] where style is Plain, Bold, Italic, or BoldItalic (e.g. “Arial,BoldItalic,12”, “Lucida Console,Plain,10”).

Example 29 creates a plot from a DSS data set and sets the Y1 axis label text to blue.

Example 29: Using AxisLabel Objects

```
from hec.script import *                        # for Plot class
from hec.hecmath import *                       # for DSS class
thePlot = Plot.newPlot()                        # create the plot
dssFile = DSS.open("J:/apps/forecast.dss")      # open the DSS file
flow = dssFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/")
                                                # read the data set
thePlot.addData(flow.getData())                 # add the data set to the
                                                plot
thePlot.showPlot()                              # show the plot
viewport0 = thePlot.getViewport(0)              # get the first viewport
yaxislabel = viewport0.getAxisLabel("Y1")       # get the Y1 axis label
yaxislabel.setForeground(“blue”)                # set the Y1 axis label text
                                                to blue
```

7.10.12 Templates

Template files saved interactively from HEC-DSSVue may be applied to plots via scripting. When saving a template interactively from the plot window via the "Save Template…" entry on the "File" menu, HEC-DSSVue:

1. Chooses the "My Documents" subdirectory of the directory specified in the USERPROFILE environment variable as the default location for the template file.
2. Appends ".template" to the end of the specified file name.

The applyTemplate(string filename) G2dDialog method requires the actual file name for the template file. To apply a template saved in the default directory, the complete template file name must be re-created as demonstrated in Example 30.

Example 30: Applying Template Saved in Default Directory

```
import os                                     # for getenv() & sep
from hec.script import *                      # for Plot class
from hec.hecmath import *                     # for DSS class
thePlot = Plot.newPlot()                      # create a new Plot
dssfile = DSS.open("C:/mydb.dss")             # open the DSS file
stage = dssfile.read("//AXEMA/STAGE/01OCT2001/1HOUR/OBS/");
                                              # read the data set
thePlot.addData(stage.getData())              # add the data set
thePlot.showPlot()                            # show the plot
templateName = "myTemplate"                   # template base name
templateFileName =                 \          # re-create the file name
  os.getenv("userprofile")         \
  + os.sep                         \
  + "My Documents"                 \
  + os.sep                         \
  + templateName                   \
  + ".template"
thePlot.applyTemplate(templateFileName)       # apply the template
```

7.11 Plot Component Properties

The following tables are the valid values to be used when calling plot related functions that take a color (setBackground(string color), etc…), an alignment (setAlignment()), a rotation (setRotation()), a fill pattern (setFillPattern()), a fill type (setFillType()), a line style (setLineStyle()), a step style (setLineStepStyle()).or a symbol type (setSymbolType()).

7.11.1 Colors

Colors can be specified either by a String or by a java.awt.Color object. If setting a color through the use of a String object the String can either be a standard color name (i.e. "darkred") or an RGB string (i.e. "255,20,20"). Table 7.22 lists standard color names.

Table 7.22 Standard Colors

black
blue
cyan
darkblue
darkcyan
darkgray
darkgreen
darkmagenta
darkorange
darkpink
darkpurple
darkred
darkyellow
gray
green
lightblue
lightcyan
lightgray
lightgreen
lightmagenta
lightorange
lightpink
lightpurple
lightred
lightyellow
magenta
orange
pink
purple
red
white
yellow

7.11.2 Alignment

Table 7.23 lists supported text alignments.

Table 7.23 Alignment

Left	Center	Right

7.11.3 Positions

Table 7.24 lists supported text positions.

Table 7.24 Positions

Above	Center	Below

7.11.4 Rotation

Table 7.25 lists supported text rotation values.

Table 7.25 Rotations

0	90	180	270

7.11.5 Fill Patterns

Table 7.26 lists supported fill patterns.

Table 7.26 Fill Patterns

Solid	Horizontal	Vertical
Cross	FDiagonal	BDiagonal
Diagonal Cross	None	

7.11.6 Fill Types

Table 7.27 lists supported fill types.

Table 7.27 Fill Types

None	Above	Below

7.11.7 Line Styles

Table 7.28 lists supported line style values.

Table 7.28 Line Styles

Solid	Dash	Dot
Dash Dot	Dash Dot-Dot	

7.11.8 Step Style

Table 7.29 lists supported step style values.

Table 7.29 Step Styles

Normal	Step	cubic

7.11.9 Symbol Types

Table 7.30 lists supported symbol type values.

Table 7.30 Symbol Types

Asterisk	Backslash	Backslash Square
Circle	Diamond	Forwardshlash
Forwardslash Square	Hash	Hash Diamond
Hash Square	Hash Triangle	Hash Triangle2
Hourglass	Open Circle	Open Diamond
Open Hourglass	Open Square	Open Triangle
Open Triangle2	Pipe	Pipe Diamond
Pipe Square	Plus	Plus Circle
Plus Diamond	Plus Square	Square
Triangle	Triangle2	X
X Circle	X Square	X Triangle
X Triangle2		

7.12 Tables

Tables allow you to view data in a vertical scrolling window that shows the ordinates, the dates and times and the values for the selected data sets.

7.12.1 Tabulate Class

```
Tabulate.newTable()
Tabulate.newTable(string title)
```

The Tabulate class in the hec.script module is used to create a new Table dialog. It contains two functions to create a Table dialog, each of which returns as a TableFrame object.

Example 31 illustrates creation of a table.

Example 31: Creating a Table

```
from hec.script import *
myTable = Tabulate.newTable()
```

or

```
from hec.script import *
myTable = Tabulate.newTable("Elevation vs Flow")
```

7.12.2 TableFrame Class

Table 7.31 describes **TableFrame** methods.

Table 7.31 - TableFrame Methods

Method	Returns	Description
addData(DataContainer dc)	integer	Adds Data Set to the table.
export()	None	Brings up the Table Export Options dialog.
export(string fileName, TableExportOptions options)	None	Exports table to specified file with specified options
exportAsHTML(string fileName)	None	Exports table in HTML format to specified file with no title and elements indented with tabs
exportAsHTML(string fileName, string title, string indent)	None	Exports table in HTML format to specified file with specified title and indentation string
exportAsXML(string fileName)	None	Exports table in XML format to specified file with no title and elements indented with tabs

Method	Returns	Description
exportAsXML(string fileName, string title, string indent)	None	Exports table in XML format to specified file with specified title and indentation string
getCellBackground(integer row, integer column)	Color	Returns the background color of the specified cell as a Color
getCellBackgroundString(integer row, integer column)	string	Returns the background color of the specified cell as a string
getCellForeground(integer row, integer column)	Color	Returns the foreground color of the specified cell as a Color
getCellForegroundString(integer row, integer column)	string	Returns the foreground color of the specified cell as a string
getColumn(DataContainer dc)	integer	Returns the number of the column that contains the specified data, if the parameter is time series data, or the number of the column that contains the x-ordinates if the parameter is paired data
getColumn(string header)	integer	Returns the number of the column that has the specified header text. Line breaks in the header text are specified as "\n"
getColumnBackground(integer column)	Color	Returns the background color of the specified column as a Color
getColumnBackgroundString(integer column)	string	Returns the background color of the specified column as a string
getColumnForeground(integer column)	Color	Returns the foreground color of the specified column as a Color
getColumnForegroundString(integer column)	string	Returns the boreground color of the specified column as a string
getColumnHeaderBackgroung(integer column)	Color	Returns the background color of the header of the specified column as a Color
getColumnHeaderBackgroungString(integer column)	string	Returns the background color of the header of the specified column as a string
getColumnHeaderFontString(integer column)	string	Returns the font of the header of the specified column as a string[1].

Method	Returns	Description
getColumnHeaderForegroung(integer column)	Color	Returns the foreground color of the header of the specified column as a Color
getColumnHeaderForegroungString(integer column)	string	Returns the foreground color of the header of the specified column as a string
getColumnLabel(integer colNum)	string	Returns the column header text for the specified column
getColumnLabels()	list of strings	Returns the column header text for all columns
getColumnWidth(integer colNum)	integer	Returns the width of the specified column in pixels
getColumnWidths()	list of integers	Returns a list of all the column widths in pixels
getCommasState()	0/1	Get whether the commas are shown
getDateTimeAsTwoColumnsState()	0/1	Get whether date/time columns are shown as 1 or 2 columns in the table
getExportString(TableExportOptions options)	string	Returns a string representation of the table exported according to the specified options
getHeight()	integer	Return the height of the table in screen coordinates.
getHTMLExportString()	string	Returns a string representation of the table exported in HTML format with no title and elements indented with tabs
getTableTitle()	G2dTitle	Returns the title of the TableFrame object as a G2dTitle object.
getTableTitleText()	string	Returns the title of the TableFrame object as a string.
getHTMLExportString(string title, string indent)	string	Returns a string representation of the table exported in HTML format with the specified title and indentation string
getLocation()	Point	Returns the location of the table in screen coordinates[2].
getRowBackground(integer row)	Color	Returns the background color of the specified row as a Color
getRowBackgroundString(integer row)	string	Returns the row background color of the specified column as a string

Method	Returns	Description
getRowForeground(integer row)	Color	Returns the foreground color of the specified row as a Color
getRowForegroundString(integer row)	string	Returns the row foreground color of the specified column as a string
getSize()	Dimension	Return the dimensions of the table in screen coordinates.
getWidth()	integer	Return the width of the table in screen coordinates.
GetXMLExportString()	string	Returns a string representation of the table exported in XML format with no title and elements indented with tabs
getXMLExportString(string title, string indent)	string	Returns a string representation of the table exported in XML format with the specified title and indentation string
hide()	None	Hide the table
iconify()	None	Minimize (iconify) the table
maximize()	None	Maximize the table
minimize()	None	Minimize (iconify) the table
print()	None	Display the print table
restore()	None	Restore the table from a minimized or maximized state
setCellBackgound(integer row, integer column, string color)	None	Sets the background color of the specified cell to the specified color
setCellForeground(integer row, integer column, string color)	None	Sets the foreground color of the specified cell to the specified color
setColumnBackgound(integer column, string color)	None	Sets the background color of the specified column to the specified color
setColumnForeground(integer column, string color)	None	Sets the foreground color of the specified column to the specified color
setColumnHeaderBackgound(integer column, string color)	None	Sets the background color of the header of the specified column to the specified color
setColumnHeaderFont(integer column, string font)	None	Sets the font of the header of the specified column to the specified font[1].

Method	Returns	Description
setColumnHeaderForeground(integer column, string color)	None	Sets the foreground color of the header of the specified column to the specified color
setColumnLabel(integer column, string label)	None	Sets the column header text of the specified column to the specified label
setColumnLabels(list labels)	None	Sets the column header text of all columns to the labels specified in the list of strings
setColumnPrecision(integer colNum, integer precision)	None	Sets the number of decimal places to display for the specified column
setColumnWidth(integer colNum, integer width)	None	Sets the width of the specified column in pixels
setColumnWidths(list widths)	None	Sets the width in pixels of all the columns to those specified in the parameter (list of integers)
setCommasState(0/1 showCommas)	None	Set state to show commas or not
setDateTimeAsTwoColumnsState(integer showDateTimeAs2Columns)	None	Set whether date/time columns should show as 1 or 2 columns in the table
setLocation(integer x, integer y)	None	Sets the location of the table in screen coordinates[2].
setSize(int width, int height)	None	Sets the size of the table in screen coordinates.
setRowBackgound(integer row, string color)	None	Sets the background color of the specified row to the specified color
setRowForeground(integer row, string color)	None	Sets the foreground color of the specified row to the specified color
setTableTitleText(string title)	None	Sets the title of the TableFrame object.
showTable()	None	Show the table

[1] Fonts are specified as *name*[,*style*[,*size*]] where *style* is Plain, Bold, Italic, or BoldItalic (e.g. "Arial,BoldItalic,12", "Lucida Console,Plain,10").

[2] The coordinate system used is a graphics coordinate system, where X increases to the right and Y increases downward from the origin (0,0) which is located in the top left corner of the display. Locations set or retrieved refer to the top left corner of the plot in reference to this coordinate system.

7.13 TableExportOptions Class

TableExportOptions objects hold complete descriptions of options for exporting tables to fixed-column-width or column-delimited text files. TableExportOptions objects have fields that are settable by the user to create marker lines of various styles. New TableExportOptions objects are created by calls to TableExportOptions() (e.g. myOptions = TableExportOptions()). Table 7.32 describes TableExportOptions fields.

Table 7.32 – TableExportOptions Fields

Field	Type	Description	Default
delimiter	string (one character)	Placed between fields if fixedWidthCol is Constants.FALSE	'\t' (tabcharacter)
quotedStrings	0/1	Specifies whether to enclose text in quotes	Constants.FALSE
title	string	Title of the table	None
fixedWidthCols	0/1	Specifies whether fields are exported as fixed-width columns or fields separated by delimiter	Constants.FALSE
columnHeader	0/1	Specifies whether the column headers are to be exported	Constants.TRUE
rowHeader	0/1	Specifies whether the row headers are to be exported	Constants.FALSE
gridLines	0/1	Specifies whether the table will be exported with text "lines" between the rows and columns	Constants.FALSE

Example 32 creates a table from two DSS data sets and exports the table as a comma-separated-value text file.

Example 32: Filling, Displaying and Exporting a Table

```
from hec.hecmath import *                  # for DSS
from hec.script import *                   # for Tabulate
file = "j:/apps/forecast.dss"              # specify the DSS file
dssfile = DSS.open(file)                   # open the file
# read 2 paths
stage = dssfile.read("//AXEMA/STAGE/01OCT2001/1HOUR/OBS/")
flow = dssfile.read("//AXEMA/FLOW/01OCT2001/1HOUR/OBS/")
theTable = Tabulate.newTable()             # create the table
theTable.setTitle("Test Table")            # set the table title
theTable.setLocation(5000,5000)            # set the location of the table off
                                           the screen
flowDC = flow.getData()                    # extract data containers
stageDC = stage.getData()
theTable.addData(flowDC)                   # add the data
theTable.addData(stageDC)
theTable.showTable()                       # show the table
flowCol = theTable.getColumn(flowDC)       # adjust columns
stageCol = theTable.getColumn(stageDC)
flowWidth = theTable.getColumnWidth(flowDC)
stageWidth = theTable.getColumnWidth(stageDC)
theTable.setColumnPrecision(flowCol, 0)
theTable.setColumnPrecision(stageCol, 2)
theTable.setColumnWidth(flowCol, flowWidth - 10)
theTable.setColumnWidth(stageCol, stageWidth + 10)
opts = TableExportOptions()                # get new export options
opts.delimiter = “,”                       # delimit with commas
opts.title = “My Table”                    # set the title
fileName = “c:/table.txt”                  # set the output file name
theTable.export(fileName, opts)            # export to the file
theTable.close()                           # close
```

7.14 Deprecated Items

Deprecated items are items whose use is discouraged. Deprecated items are currently supported, but may not be in a future version of the software. As such, they should be removed and replaced with the recommended items at the earliest convenience.

Table 7. 33 lists currently deprecated items.

Table 7. 33– Deprecated Items

Item	Comment
addDisplayObject(DisplayObject) G2dDialog method	Replaced with **addData(DataContainer)** G2dDialog method
addDisplayObject(DisplayObject) TableFrame method	Replaced with **addData(DataContainer)** TableFrame method
areMajorTicLabelsDrawn() AxisTics method	Replaced with **areMajorTicLabelsVisible()** AxisTics method
areMajorTicsDrawn() AxisTics method	Replaced with **areMajorTicsVisible()** AxisTics method
areMinorTicLabelsDrawn() AxisTics method	Replaced with **areMinorTicLabelsVisible()** AxisTics method
areMinorTicsDrawn() AxisTics method	Replaced with **areMinorTicsVisible()** AxisTics method
areSymbolsDrawn() G2dLine method	Replaced with **areSymbolsVisible()** G2dLine method
createDisplayObject(HecMath) function	Replaced with **getData()** HecMath method
DisplayUtilities import module	No longer needed
getActMax() Axis method	Replaced with **getViewMax()** Axis method
getActMin() Axis method	Replaced with **getViewMin()** Axis method
getAutoSkipSymbols() G2dLine method	Replaced with **areSymbolsAutoInterval()** G2dLine method
getGridColorX() Viewport method	Replaced with **getMajorGridXColor()** Viewport method
getGridColorY() Viewport method	Replaced with **getMajorGridYColor()** Viewport method
getGridXColorString() Viewport method	Replaced with **getMajorGridXColorString()** Viewport method

Item	Comment
getGridYColorString() Viewport method	Replaced with **getMajorGridYColorString()** Viewport method
getMajorXGridWidth() Viewport method	Replaced with **getMajorGridXWidth()** Viewport method
getMajorYGridWidth() Viewport method	Replaced with **getMajorGridYWidth()** Viewport method
getMax() Axis method	Replaced with **getScaleMax()** Axis method
getMin() Axis method	Replaced with **getScaleMin()** Axis method
getMinorXGridWidth() Viewport method	Replaced with **getMinorGridXWidth()** Viewport method
getMinorYGridWidth() Viewport method	Replaced with **getMinorGridYWidth()** Viewport method
getReversed() Axis method	Replaced with **isReversed()** Axis method
getSymbolOffset() G2dLine method	Replaced with **getFirstSymbolOffset()** G2dLine method
getSymbolSkipInterval() G2dLine method	Replaced with **getSymbolInterval()** and **getSymbolSkipCount()** G2dLine methods
isBackgroundDrawn() G2dLabel method	Replaced with **isBackgroundVisible()** G2dLabel method
isBackgroundDrawn() Viewport method	Replaced with **isBackgroundVisible()** Viewport method
isBorderDrawn() G2dLabel method	Replaced with **isBorderVisible()** G2dLabel method
isBorderDrawn() Viewport method	Replaced with **isBorderVisible()** Viewport method
isLineDrawn() G2dLine method	Replaced with **isLineVisible()** G2dLine method
isMajorGridXDrawn() Viewport method	Replaced with **isMajorGridXVisible()** Viewport method
isMajorGridYDrawn() Viewport method	Replaced with **isMajorGridYVisible()** Viewport method
isMinorGridXDrawn() Viewport method	Replaced with **isMinorGridXVisible()** Viewport method
isMinorGridYDrawn() Viewport method	Replaced with **isMinorGridYVisible()** Viewport method
isTitleDrawn() G2dTitle method	Replaced with **isPlotTitleVisible()** G2dDialog method

Item	Comment
setAutoSkipSymbolsOff() G2dLine method	Replaced with **setSymbolsAutoInterval(0/1)** G2dLine method
setAutoSkipSymbolsOn() G2dLine method	Replaced with **setSymbolsAutoInterval(0/1)** G2dLine method
setComputeMajorTicsOff() Axis method	Replaced with **setComputeMajorTics(0/1)** Axis method
setComputeMajorTicsOn() Axis method	Replaced with **setComputeMajorTics(0/1)** Axis method
setComputeMinorTicsOff() Axis method	Replaced with **setComputeMinorTics(0/1)** Axis method
setComputeMinorTicsOn() Axis method	Replaced with **setComputeMinorTics(0/1)** Axis method
setDrawBackgroundOff() Viewport method	Replaced with **setBackgroundVisible(0/1)** Viewport method
setDrawBackgroundOn() Viewport method	Replaced with **setBackgroundVisible(0/1)** Viewport method
setDrawBackroundOff() G2dLabel method	Replaced with **setBackgroundVisible(0/1)** G2dLabel method
setDrawBackroundOn() G2dLabel method	Replaced with **setBackgroundVisible(0/1)** G2dLabel method
setDrawBorderOff() G2dLabel method	Replaced with **setBorderVisible(0/1)** G2dLabel method
setDrawBorderOff() Viewport method	Replaced with **setBorderVisible(0/1)** Viewport method
setDrawBorderOn() G2dLabel method	Replaced with **setBorderVisible(0/1)** G2dLabel method
setDrawBorderOn() Viewport method	Replaced with **setBorderVisible(0/1)** Viewport method
setDrawLineOff() G2dLine method	Replaced with **setLineVisible(0/1)** G2dLine method
setDrawLineOn() G2dLine method	Replaced with **setLineVisible(0/1)** G2dLine method
setDrawMajorTicLabelsOff() AxisTics method	Replaced with **setMajorTicLabelsVisible(0/1)** Axis method

Item	Comment
setDrawMajorTicLabelsOn() AxisTics method	Replaced with **setMajorTicLabelsVisible(0/1)** Axis method
setDrawMajorTicsOff() AxisTics method	Replaced with **setMajorTicsVisible(0/1)** Axis method
setDrawMajorTicsOn() AxisTics method	Replaced with **setMajorTicsVisible(0/1)** Axis method
setDrawMinorTicLabelsOff() AxisTics method	Replaced with **setMinorTicLabelsVisible(0/1)** Axis method
setDrawMinorTicLabelsOn() AxisTics method	Replaced with **setMinorTicLabelsVisible(0/1)** Axis method
setDrawMinorTicsOff() AxisTics method	Replaced with **setMinorTicsVisible(0/1)** Axis method
setDrawMinorTicsOn() AxisTics method	Replaced with **setMinorTicsVisible(0/1)** Axis method
setDrawMajorXGridOff() Viewport method	Replaced with **setMajorGridXVisible(0/1)** Viewport method
setDrawMajorXGridOn() Viewport method	Replaced with **setMajorGridXVisible(0/1)** Viewport method
setDrawMajorYGridOff() Viewport method	Replaced with **setMajorGridYVisible(0/1)** Viewport method
setDrawMajorYGridOn() Viewport method	Replaced with **setMajorGridYVisible(0/1)** Viewport method
setDrawMinorXGridOff() Viewport method	Replaced with **setMinorGridXVisible(0/1)** Viewport method
setDrawMinorXGridOn() Viewport method	Replaced with **setMinorGridXVisible(0/1)** Viewport method
setDrawMinorYGridOff() Viewport method	Replaced with **setMinorGridYVisible(0/1)** Viewport method
setDrawMinorYGridOn() Viewport method	Replaced with **setMinorGridYVisible(0/1)** Viewport method
setDrawSymbolsOff() G2dLine method	Replaced with **setSymbolsVisible(0/1)** G2dLine method
setDrawSymbolsOn() G2dLine method	Replaced with **setSymbolsVisible(0/1)** G2dLine method

Item	Comment
setGridColorX(string) Viewport method	Replaced with **seGridXColor(string)** Viewport method
setGridColorY(string) Viewport method	Replaced with **setGridYColor(string)** Viewport method
setMajorXGridWidth(floating-point) Viewport method	Replaced with **setMajorGridXWidth(floating-point)** Viewport method
setMajorYGridWidth(floating-point) Viewport method	Replaced with **setMajorGridYWidth(floating-point)** Viewport method
setMinorXGridWidth(floating-point) Viewport method	Replaced with **setMinorGridXWidth(floating-point)** Viewport method
setMinorYGridWidth(floating-point) Viewport method	Replaced with **setMinorGridYWidth(floating-point)** Viewport method
setReversedOff() Axis method	Replaced with **setReversed(0/1)** Axis method
setReversedOn() Axis method	Replaced with **setReversed(0/1)** Axis method
setSymbolOffset(integer) G2dLine method	Replaced with **setFirstSymbolOffset(integer)** G2dLine method
setSymbolSkipInterval(integer) G2dLine method	Replaced with **setSymbolInterval(integer)** and **setSymbolSkipCount(integer)** G2dLine methods
setTitleDrawOff() G2dTitle method	Replaced with **setPlotTitleVisible(0/1)** G2dDialog method
setTitleDrawOn() G2dTitle method	Replaced with **setPlotTitleVisible(0/1)** G2dDialog method
zoomIn(floating-point, floating-point) Axis method	Replaced with **setScaleLimits(floating-point, floating-point)** Axis method

7.15 Math Functions

Math functions are accessible through the general class called **HecMath**. **HecMath** objects hold data sets and allow you to perform mathematical operations on them. They can also be passed to plots and tables to display the data. A HecMath object is either a TimeSeriesMath object or a PairedDataMath object, which handle time series and paired data sets, respectively.

Before using PairedDataMath methods, be sure to read the description for the setCurve Method. Paired data sets may contain multiple curves. The setCurve method provides user control over which paired data curve is operated upon by subsequent function calls.

7.15.1 Absolute Value

`abs()`

Derive a new time series or paired data set from the absolute value of values of the current data set. For time series data, missing values are kept as missing. For paired data sets, use the setCurve method to first select the paired data curve(s).

See also: setCurve().

Parameters: Takes no parameters.

`Example:  NewDataSet = dataSet.abs()`

Returns: A new HecMath object of the same type as the current object.

7.15.2 Accumulation (Running)

`accumulation``()`

Derive a new time series by computing a running accumulation of the current time series.

For time points in which the current time series value are missing, the value in the accumulation time series remains constant (same as the accumulated value at the last valid point location).

Parameters: Takes no parameters.

`Example:  NewTimeSeries = timeSeries.accumulation``()`

Returns: A new TimeSeriesMath object.

7.15.3 Add a Constant

`add(floating-point constant)`

Add the value **constant** to all valid values in the current time series or paired data set. For time series data, missing values are kept as missing.

For paired data, **constant** is added to y-values only. Use the setCurve method to first select the paired data curve(s).

See also: add(HecMath dataSet)

setCurve()

Parameters: `constant` - A floating-point value.

Example: `newDataSet = dataSet.add(2.5)`

Returns: A new HecMath object of the same type as the current object.

7.15.4 Add a Data Set

`add(TimeSeriesMath tsData)`

Add the values in the data set **tsData** to the values in the current data set. The function only applies to time series data sets.

When adding one time series data set to another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be added. Times in the current time series data set that cannot be matched with times in the second data set are set to missing. Values in the current data set that are missing are kept as missing. Either or both data sets may be regular or irregular interval time series.

This function will not merge data sets. Use the mergeTimeSeries (for time series data sets) or the mergePairedData (for paired data sets) functions for this purpose.

See also: add(floating-point constant)

mergeTimeSeries(TimeSeriesMath)

mergePairedData(PairedDataMath)

Parameters: `tsData` - A TimeSeriesMath object.

Example: `newTsData = tsData.add(otherTsData)`

Returns: A new TimeSeriesMath object.

7.15.5 Apply Multiple Linear Regression Equation

```
applyMultipleLinearRegression(string startTimeString,
            string endTimeString,
            sequence tsDataSetSequence,
            floating-point minimumLimit,
            floating-point maximumLimit)
```

Apply the regression coefficients contained in the current paired data set to the array of time series data sets in **tsDataSetSequence** to develop a new time series data set. The applyMultipleLinearRegression function applies the multiple linear regression coefficients computed with the multipleLinearRegression function (see section 7.15.45).

For the general linear regression equation, a dependent variable, Y, may be computed from a set independent variables, Xn:

Y = B0 + B1*X1 + B2*X2 + B3*X3

where Bn are linear regression coefficients.

For time series data sets, an estimate of the original time series data set values may be computed from a set of independent time series data sets using regression coefficients such that:

TsEstimate(t) = B0 + B1*TS1(t) + B2*TS2(t) + … + Bn*TSn(t)

where Bn are the set of regression coefficients and TSn are the time series data sets contained in **tsDataSetSequence**.

The number of regression coefficients in the current PairedDataMath object must be one more than the number of independent time series data sets in **tsDataSetSequence**. The collection of selected time series data sets must be in the same order as when the regression coefficients were computed with the multipleLinearRegression method.

All the time series data sets must be regular interval and have the same time interval. The function filters the data to determine the time period common to all time series data sets and uses only those points in the regression analysis. For any given time, if a value is missing in any time series, the value in resultant time series is set to missing.

The parameters **minimumLimit** and **maximumLimit** can be used to specify the range of valid values for the resultant data set. Values which fall outside the specified range are set to missing. **minimumLimit** or **maximumLimit** may be entered as **Constants.UNDEFINED** to ignore the minimum or maximum value check.

If **startTimeString** or **endTimeString** are blank strings, the start and end time of the resultant time series will be defined by the time period common to all time series data sets in **tsDataSetSequence**. Otherwise the time series start and end may be defined using **startTimeString** and **endTimeString** which have the usual HEC time window format (e.g. "01JAN2001 1400").

Names, parameter type and unit labels for the new time series data set are copied over from the first time series data set in **tsDataSetSequence**. The F-part in the new data set is set to "COMPUTED."

Parameters:

`startTimeString` - A string containing an HEC time (e.g. "01JAN2001 1400") specifying the start time of the resultant time series data set. May be blank (" ").

`endTimeString` - A string containing an HEC time (e.g. "01JAN2001 1400") specifying the ending time of the resultant time series data set. May be blank (" ").

`tsDataSetSequence` – Sequence of TimeSeriesMath objects. Must all be regular interval and have the same time interval.

`minimumLimit` – A floating-point value specifying the minimum valid value in the resultant time series data set. Set to Constants.UNDEFINED to ignore this option.

`maximumLimit` – A floating-point value specifying the maximum valid value in the resultant time series data set. Set to Constants.UNDEFINED to ignore this option.

Example:

```
newTsData =
pairedData.applyMultipleLinearRegression(
      "01Jan2000 0000",
      "31Dec2000 2300",
      (tsData1, tsData2, tsData3),
      Constants.UNDEFINED,
      Constants.UNDEFINED)
```

Returns: A new regular interval TimeSeriesMath object.

Generated Exceptions: `Throws a` HecMathException if the number of data sets in **tsDataSetSequence** is not equal to the number of regression coefficients -1, or if the data sets in **tsDataSetSequence** are not regular interval time series data sets with the same interval time.

7.15.6 Centered Moving Average Smoothing

```
centeredMovingAverage(integer numberToAverageOver,
      boolean onlyValidValues,
      boolean useReduced)
```

Derive a new time series from the centered moving average of **numberToAverageOver** values in the current time series. **numberToAverageOver** must be an odd integer greater than 2.

If **onlyValidValues** is set to true, then if any points in the averaging interval are missing, the point in the new time series is set to missing. If **onlyValidValues** is set to false and missing values are contained in the averaging interval, a smoothed point is still computed using the remaining valid values in the interval. If there are no valid values in the averaging interval, the point is set to missing.

If **useReduced** is set to true, then centered moving average points can still be computed at the beginning and end of the time series, even if there are less than **numberToAverageOver** values in the averaging interval. If **useReduced** is set to false, then the first and last **numberToAverageOver**/2 points of the resultant time series are set to missing.

Parameters:

numberToAverageOver – An integer containing the number of values to average over for computing the centered moving average. Must be odd and greater than 2.

onlyValidValues – Either Constants.TRUE, or Constants.FALSE, specifying whether all values in the averaging interval must be valid for the computed point in the new time series to be valid.

useReduced – Either Constants.TRUE, or Constants.FALSE, specifying whether to allow points at the beginning and end of the resultant time series to be computed from a reduced (less than numberToAverageOver) set of points.

Example:

```
avgData = tsData.centeredMovingAverage (5,
      Constants.TRUE,
      Constants.TRUE)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws a HecMathException if the numberToAverageOver is less than 3 or not odd.

7.15.7 Conic Interpolation from Elevation/Area Table

```
conicInterpolation(  TimeSeriesMath tsData,
      string inputType,
      string outputType,
      floating-point storageScaleFactor )
```

Use the conic interpolation table in the current paired data set to develop a new time series data set from the interpolation of **tsData**.

The current paired data should be an Elevation-Area table. However, the first data pair are the initial conic depth, and the storage value at the first elevation in the table. If the initial conic depth is undefined, the function will calculate a value. Elevation-Area values in the table must be in ascending order.

tsData is either a time series of reservoir elevation or storage. The type is specified by setting **inputType** as "S(TORAGE)" or "E(LEVATION)." The desired output time series type is similarly set using **outputType**. The valid settings for **outputType** are "S(TORAGE)", "E(LEVATION)" or "A(REA)." **inputType** and **outputType** must not be the same.

storageScaleFactor is an optional parameter used to scale input (by multiplying) and output (by dividing) storage values. For example, if the area in the conic interpolation table is expressed in sq.ft., **storageScaleFactor** could be set to 43560. to convert the storage output to acre-ft.

Parameter type in the new time series is set according to **outputType**. If the output time series values are elevation, the time series units are set to the paired data x-units label. If the output time series values are area, the time series units are set to the paired data y-units label. If the output is storage, the units are not set and should be set by the user with the setUnits function.

See also: setUnits().

Parameters:

`tsData` – A TimeSeriesMath object representing elevation or storage.

`inputType` – A string specifying the parameter type for the input time series, either "S(TORAGE)" or "E(LEVATION)." Only the first character of the string is interpreted by the function.

`outputType` – A string specifying the parameter type for the output time series, either "S(TORAGE)", "E(LEVATION)" or "A(REA)." Only the first character of the string is interpreted by the function.

`storageScaleFactor` – A floating-point number used to scale input (by multiplying) and output (by dividing) storage values.

Examples:

```
tsStorage =
      conicElevAreaCurve.conicInterpolation(
      tsElev,
      “Elevation”,
      “Storage”,
      1.0)


tsArea =
      conicElevAreaCurve.conicInterpolation(
      tsElev,
      “Elevation”,
      “Area”,
      1.0)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: `Throws an` hec.hecmath.HecMathException if **inputType** or **outputType** cannot be interpreted as one of the allowed values; if **inputType** and **outputType** are the same parameters; if values in the conic interpolation table are not in ascending order.

7.15.8 Convert Values to English Units

`convertToEnglishUnits()`

Perform unit conversion of data values and unit labels in the current time series or paired data set from Metric (SI) units to English units. Determination of the unit system will be based upon the current units labels and parameter types. If the data units are already in English units or the unit system cannot be determined, no conversion occurs.

For paired data, both x and y values are converted. For time series data, missing values remain missing.

See also: convertToMetricUnits(), isEnglish(), isMetric().

Parameters: Takes no parameters

Example: `englishDataSet = siDataSet.convertToEnglishUnits()`

Returns: A HecMath object of the same type as the current object.

7.15.9 Convert Values to Metric (SI) Units

convertToMetricUnits()

Perform unit conversion of data values and unit labels in the current time series or paired data set from English units to Metric (SI) units. Determination of the unit system will be based upon the current units labels and parameter types. If the units are already in Metric units or the unit system cannot be determined, no conversion occurs.

For paired data, both x and y values are converted. For time series data, missing values remain missing.

See also: convertToEnglishUnits(), isEnglish(), isMetric().

Parameters: Takes no parameters

```
Example:  siDataSet = englishDataSet.convertToMetricUnits()
```

Returns: An HecMath object of the same type as the current object.

7.15.10 Correlation Coefficients

`correlationCoefficients``(TimeSeriesMath tsData)`

Computes the linear regression and other correlation coefficients between data in the current time series and `tsData`. Values in the current time series and `tsData` are matched by time to form data pairs for the correlation analysis. The data sets may be either regular or irregular time interval data.

The correlations statistics computed by the function are:

Number of valid values

Regression constant

Regression coefficient

Determination coefficient

Standard error of regression

Determination coefficient adjusted for degrees of freedom

Standard error adjusted for degrees of freedom

These values are contained in a LinearRegressionStatistics object.

The current TimeSeriesMath object forms the values of the independent variable (x-values), while values of the second time series comprise the dependent variable (y-values). The linear regression coefficients thus express how values in the second data set can be derived from values in the primary data set:

TS2(t) = a + b * TS1(t)

where "a" is the regression constant and "b" the regression coefficient.

See also: LinearRegressionStatistics.

Parameters: tsData - A TimeSeriesMath object that forms the dependent variable for the regression analysis.

Example:

```
    linearRegressionData =
          tsData.correlationCoefficients(otherTsData)
```

Returns:A LinearRegressionStatistics object holding the correlation data.

Generated Exceptions: `Throws an` hec.hecmath.HecMathException if the times in the current time series do not exactly match times in tsData.

7.15.11Cosine Trigonometric Function

`cos()`

Derive a new time series or paired data set from the cosine of values of the current data set. The resultant data set values are in radians. For time series data, missing values are kept as missing.

For paired data sets, use the setCurve (see sections 7.15.61 and 7.15.62) function to first select the paired data curve (or all curves) to apply the function. By default the function is applied to all paired data curves.

See also: setCurve().

Parameters: Takes no parameters

Returns: A HecMath object of the same type as the current object.

7.15.12 Cyclic Analysis (Time Series)

cyclicAnalysis()

Derive a set of cyclic statistics from the current regular interval time series data set. The time series data set must have a time interval of "1HOUR", "1DAY" or "1MONTH." The function sorts the time series values into statistical "bins" relevant to the time interval. Values for the 1HOUR interval data are sorted into 24 bins representing the hours of the day, 0100 to 2400. The 1DAY interval data is apportioned to 365 bins for the days of the year. The 1MONTH interval data is sorted into 12 bins for the months of the year.

The format of the resultant data sets is as a "pseudo" time series for the year 3000. For example, the cyclic analysis of one month of hourly interval data will produce pseudo time series data sets having 24 hourly values for the day January 1, 3000. If the statistical parameter is the "maximum" value, then the 24 values represent the maximum value occurring at that hour of the day in the current time series. The cyclic analysis of daily interval data will produce pseudo time series data sets having 365 daily values for the year 3000. The cyclic analysis of monthly interval data will result in pseudo time series data sets having 12 monthly values for the year 3000.

Fourteen pseudo time series data sets are derived by the cyclic analysis function for the following statistical parameters:

- Number of values processed for each time interval
- Maximum value
- Time of maximum value
- Minimum value
- Time of minimum value
- Average value
- Probability exceedence percentiles for 5%, 10%, 25%, 50% (median value), 75%, 90%, and 95%
- Standard deviation

The 14 pseudo time series of cyclic statistics are returned by the function as an array of time series data sets. The parameter part of the record path for each time series is modified to indicate the type of the statistical parameter. For a flow record, the parameter "FLOW" would become "FLOW-MAX" for the maximum values statistics, "FLOW-P5" for the 5% percentile statistics, etc.

Parameters: Takes no parameters

```
Example: cyclicData = tsData.cyclicAnalysis()
```

Returns: A sequence of 14 TimeSeriesMath objects, each of which is a pseudo time series data sets representing a statistical parameter.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the time series is not regular interval or does not have a time interval of "1HOUR", "1DAY", or "1MONTH".

7.15.13 Decaying Basin Wetness Parameter

```
decayingBasinWetnessParameter( TimeSeriesMath tsPrecip,
      floating-point decayRate )
```

Compute a time series of decaying basin wetness parameters from the regular interval time series data set of incremental precipitation, **tsPrecip,** by:

TSResult(t) = Rate * TSResult(t-1) + TSPrecip(t)

where Rate is **decayRate**, and 0 < Rate < 1.

The first value of the resultant time series data set, TSResult(1), is set to the first value in the current time series data set. The current time series data set can be the same time series data set as **tsPrecip**. Missing values in the precipitation time series are taken as zero when applying the above equation.

Parameters:

tsPrecip – A regular interval TimeSeriesMath object representing precipitation.

decayRate – a floating-point number in the range 0 < decayRate < 1.

Example:

```
      tsWetness =
            tsPrecip.decayingBasinWetnessParameter(
            tsPrecip,
            0.87)
```

Returns: A new TimeSeriesMath object.

7.15.14 Divide by a Constant

```
divide(floating-point constant)
```

Divide all valid values in the current time series or paired data set by the value **constant**. For time series data, missing values are kept as missing.

For paired data, **constant** divides the y-values only. Use the setCurve method to select the paired data curve(s).

See also: divide(TimeSeriesMath tsData); setCurve()**.**

Parameters:

constant - A floating-point value to divide the values in the current data set (cannot be zero).

Example: newDataSet = dataSet.divide(1.1)

Returns: A new HecMath object of the same type as the current object.

7.15.15 Divide by a Data Set

`divide(TimeSeriesMath tsData)`

Divide valid values in the current data set by the corresponding values in the data set **tsData**. Both data sets must be time series data sets.

When dividing one time series data set by another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be divided. Times in the current time series data set that cannot be matched with times in the second data set are set to missing. Values in the current data set that are missing are kept as missing. If a value in the second data set is zero or missing, the value in the resultant data set is set to missing (divide by zero not allowed). Either or both data sets may be regular or irregular interval time series.

See also: divide(floating-point constant)**.**

Parameters:

`tsData` - A time series data set.

Example: `newTsData = tsData.divide(otherTsData)`

Returns: A new TimeSeriesMath object.

7.15.16 Estimate Values for Missing Precipitation Data

`estimateForMissingPrecipValues(integer maxMissingAllowed)`

Linearly interpolate estimates for missing values in the current regular or irregular interval time series data set. The current data set is expected to be cumulative precipitation and the data must be of type "INST-CUM". Use the estimateForMissingValues method for filling missing values in other types of time series data.

The rules used for interpolation of missing cumulative precipitation data are:

- If the values bracketing the missing period are increasing with time, only interpolate if the number of successive missing values does not exceed the value of **maxMissingAllowed**.
- If the values bracketing the missing period are decreasing with time, do not estimate for any missing values.
- If the values bracketing the missing period are equal, then estimate any number of missing values.

See also: estimateForMissingValues().

Parameters:

```
maxMissingAllowed - An integer value for the maximum
number of consecutive missing values between valid
values.
```

Example:

```
newPrecip =
      tsPrecip.estimateForMissingPrecipValues(5)
```

Returns: A new TimeSeriesMath object.

7.15.17 Estimate Values for Missing Data

`estimateForMissingValues(integer maxMissingAllowed)`

Linearly interpolate estimates for missing values in the current regular or irregular interval time series data set. Do not interpolate if the number of successive missing values exceeds **maxMissingAllowed**.

See also: estimateForMissingPrecipValues()**.**

Parameters:

```
maxMissingAllowed  - An integer value for the maximum
number of consecutive missing values allowed for
interpolation.
```

Example: newTsData = tsData.estimateForMissingValues(5)

Returns: A new TimeSeriesMath object.

7.15.18 Exponentiation Function

`exponentiation(floating-point constant)`

Derive a new time series or paired data set from the exponentiation of values in the current data set by **constant**, by:

$T2\ (i) = T1(i)^{constant}$

For time series data, values that are missing in the current time series remain missing in the new time series.

For paired data sets, use the setCurve method to first select the paired data curve(s).

See also: setCurve().

Parameters:

`constant` – a floating-point value representing the exponent.

Example: squaredDataSet = dataSet.exponentiation(2.)

Returns: A new HecMath object of the same type as the current object.

7.15.19 Extract Time Series Data at Unique Time Specification

```
extractTimeSeriesDataForTimeSpecification(
      string timeLevelString,
      string rangeString,
      boolean isInclusive,
      integer intervalWindow,
      boolean setAsIrregular )
```

Select/extract data points from the current regular or irregular interval time series data set based upon user defined time specifications. For example, the function may be used to extract from hourly interval data, the values observed every day at noon.

`timeLevelString` defines the time level/interval for extraction (year, month, day of the month, day of the week, or 24-hour time). `rangeString` defines the interval range for data extraction applicable to the time level. For example, if `timeLevelString` is "MONTH", a valid range would be "JAN-MAR". The `rangeString` variable can define a single interval value (e.g. "JAN" - select data from January only) or a beginning and ending range (e.g. "JAN-MAR" - select data for January through March). Table 7.34 shows the valid `timeLevelString` and `rangeString` values.

Table 7.34 - Valid timeLevelString and rangeString Values

timeLevelString	rangeString	Example rangeString
"YEAR"	Four-digit year value	"1938" or "1938-1945"
"MONTH"	Standard three-character abbreviation for month	"JAN" or "JAN-MAR" or "OCT-FEB"
"DAYMON(TH)"	Day of the month or "LASTDAY" string	"15" or "1-15 or "27-5" or "16-LASTDAY"
"DAYWEE(K)"	Standard three-character abbreviation for day of the week	"MON" or "SUN-TUE" or "FRI-WED"
"TIME"	Four digit 24-hour military-style clock time	"2400" or "0300-0600" or "2200-0130"

If desired, you may use one of the enumerated string constants to specify **timeLevelString**:

Year	TimeSeriesMath.LEVEL_YEAR_STRING
Month	TimeSeriesMath. LEVEL_MONTH_STRING
Day of Month	TimeSeriesMath. LEVEL_DAYMONTH_STRING
Day of Week	TimeSeriesMath.LEVEL_DAYWEEK_STRING
24-hour time	TimeSeriesMath.LEVEL_TIME_STRING

The parameter `isInclusive` determines whether the data extraction operation is either inclusive or exclusive of the specified range. For example, if `isInclusive` is "true" and the range is set to "JAN-MAR" for the "MONTH" time level, the extracted data will include all data in the months January through March for all the years of time series data. If `isInclusive` is "false" for this example, the extracted data covers the time April through December (is exclusive of the period January through March).

`intervalWindow` is only used when the `timeLevelString` is "TIME." `intervalWindow` is the minutes before and after the time of day within which the data will be extracted. `intervalWindow` effectively increases the time range at the beginning and end `intervalWindow` minutes. For example, with a rangeString of "0300" and an **`intervalWindow`** of 10, data will be extracted from the selected time series if times falls within in the period 0250 to 0310.

`setAsIrregular` defines whether the extracted data is saved as regular interval or irregular interval data. Most often the time series data formed by the extraction process will no longer be regular interval, and `setAsIrregular` should be set to "true." Setting `setAsIrregular` to "false" will force an attempt to save the data as regular interval time data.

Parameters:

`timeLevelString` – A string specifying the time level selection.

`rangeString` – A string specifying time or time range for selection. Must be consistent with timeLevelString.

`isInclusive` – Either Constants.TRUE, or Constants.FALSE, value. If true, data is extracted inclusive of the range specified by rangeString. If false, data is extracted exclusive of the range specified by rangeString.

`intervalWindow` – An integer value representing the minutes before and after the time of day within which the data will be extracted. Only applied when the timeLevelString is "TIME."

`setAsIrregular` – Either Constants.TRUE, or Constants.FALSE, value. If true, data is automatically set as irregular time interval data. If false, the function will attempt to classify the data as regular time interval data.

Example:

```
SelectedData =
      tsData.extractTimeSeriesDataForTimeSpecification(
      "DAYMONTH",
      "16-LASTDAY",
      Constants.TRUE,
      0,
      Constants.TRUE)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the function could not successfully interpret timeLevelString or rangeString.

7.15.20 Flow Accumulator Gage (Compute Period Average Flows)

flowAccumulatorGageProcessor(TimeSeriesMath tsCounts)

Derive a new time series of period-average flows from a flow accumulator type gage. The current time series is assumed to containe the accumulated flow data, while the parameter time series, **tsCounts**, is assumed to have the corresponding time series of counts. The two time series data sets must match times exactly. The two time series are combined to compute a new time series of period average flow:

$$TsNew(t) = (TsAccFlow(t) - TsAccFlow(t-1)) / (TsCount(t) - TsCount(t-1))$$

where TsAccFlow is the gage accumulated flow time series and TsCount is the gage time series of counts.

In the above equation, if TsAccFlow(t), TsAccFlow(t-1), TsCount(t) or TsCount(t-1) are missing, TsNew(t) is set to missing. The new time series is assigned the data type "PER-AVER".

Parameters:

tsCounts – A TimeSeriesMath object containing the counts for the flow accumulator gage.

Example:

```
tsPerAvgFlow =
      tsAccumFlow.flowAccumulatorGageProcessor(tsCounts)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws an hec.hecmath.HecMathException if times in the current object do not exactly match the times in **tsCounts**.

7.15.21 Forward Moving Average Smoothing

```
forwardMovingAverage(integer numberToAverageOver)
```

Derive a new time series from the forward moving average of **numberToAverageOver** values in the current time series. **numberToAverageOver** must be an integer greater than 2.

If the averaging interval contains a missing value, the smoothed value is computed from the remaining valid values in the interval. However, if there are less than 2 valid values in the interval, the value in the resultant data set is set to missing.

Parameters:

`numberToAverageOver` – An integer containing the number of values to average over for computing the forward moving average.

Example: `tsAveraged = tsData.forwardMovingAverage(4)`

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the numberToAverageOver is less than 2.

7.15.22 Generate Data Pairs from Two Time Series

```
generateDataPairs(TimeSeriesMath tsData,
      boolean sort)
```

Generate a paired data set by pairing values (by time) from the current time series data set and the time series data set **tsData**. The values of the current time series form the x-ordinates, while values from `tsData` form the y-ordinates of the resulting paired data set. The times in the two time series data sets must match exactly. If a value for a time is missing in either time series, no data value pair is formed or added to the paired data set. If **sort** is "true", data pairs in the paired data set are sorted by ascending x-value.

The units and parameter type from the current time series data set are assigned to the paired data set x-units and x-parameter type. The units and parameter type from **tsData** are assigned to the paired data set y-units and y-parameter type.

An example application of the function would be to mate a time series record of stage to one of flow to generate a stage-flow paired data set.

Parameters:

`tsData` – A TimeSeriesMath object that forms the y-ordinates of the resulting paired data set.

sort – Either Constants.TRUE, or Constants.FALSE, value. If true, sort data pairs in ascending x-value. If false, leave unsorted.

Example: ratingCurve = tsStage.generateDataPairs(tsFlow)

Returns: A PairedDataMath object with x-ordinates from the current time series, and y-ordinates from **tsData**.

Generated Exceptions: Throws an hec.hecmath.HecMathException if times from the current time series and **tsData** do not match exactly.

7.15.23 Generate a Regular Interval Time Series

```
generateRegularIntervalTimeSeries(string startTimeString,
            string endTimeString,
            string timeIntervalString,
            string timeOffsetString,
            floating-point initialValue )
```

Generate a new regular interval time series data set from scratch with times and values specified by the parameters. This is a function provided by the TimeSeriesMath module, and not an object method.

The parameters **startTimeString** and **endTimeString** are strings used to specify the beginning and ending time of the generated data set. These two parameters have the form of the standard HEC time string (e.g. "01JAN2001 0100").

The regular time interval is specified by **timeIntervalString**, and is a valid HEC time increment string (e.g. “1MIN”, “15MIN”, “1HOUR”, “6HOUR”, “1DAY”, “1MONTH”).

timeOffsetString is used to shift times in the resultant time series from the standard interval time. As an example, the offset could be used to shift times in regular hourly interval data from the top of the hour to 6 minutes past the hour. The parameter has the form "nT”, where "n" is an integer number and "T" is one of the time increments: "M(INUTES)”, "D(AYS)”, "H(OUR)”, "W(EEKS)”, "MON(THS)" or "Y(EARS)" (characters in the parenthesis are optional). For example, a time offset of 9 minutes would be expressed as "9M" or "9MIN.”

Values in the time series data set are initialized to **initialValue**.

Parameters:

startTimeString - a string specifying a standard HEC time defining the time series data start date/time.

endTimeString - a string specifying a standard HEC time defining the time series data end date/time.

timeIntervalString - a string specifying a valid DSS regular time interval which defines the time interval of the new time series.

timeOffsetString – a string specifying the offset of the new time points from the regular interval time. This string may be an empty string or None.

initialValue –a floating-point number set to the initial value for all time series points. Set to HecMath.UNDEFINED to set all values to missing.

Example:

```
newTsData =
      TimeSeriesMath.generateRegularIntervalTimeSeries(
            "01FEB2002 0100",
            "28FEB2002 2400",
            "1HOUR",
            "0M",
            100.)
```

Returns: A new regular interval TimeSeriesMath object initialized to `initialValue`. Data units and type are unset.

Generated Exceptions: `Throws an` hec.hecmath.HecMathException if time parameters cannot be successfully interpreted.

7.15.24 Get Data Container

getData()

Returns a copy of the hec.io.DataContainer for the current data set. For time series data sets, returns a hec.io.TimeSeriesContainer. For paired data sets, returns a hec.io.PairedDataContainer.

The hec.io.TimeSeriesContainer contains the time series values for a time series data set. The hec.io.PairedDataContainer contains the paired data values for a paired data set.

Parameters: Takes no parameters

Example: container = dataset.getData()

Returns: A hec.io.DataContainer.

7.15.25 Get Data Type for Time Series Data Set

getType()

Get the data type for a time series data set.

Parameters: Takes no parameters

Example: **dataSet.getType()**

Returns: A string - "INST-CUM", "INST-VAL", "PER-AVER" or "PER-CUM".

7.15.26 Get Units Label for Data Set

```
getUnits()
```

Get the units label of the current data set. For a paired data set, returns the y-units label.

Parameters: Takes no parameters

```
Example:  dataSet.getUnits()
```

Returns: A string.

7.15.27 Interpolate Time Series Data at Regular Intervals

```
interpolateDataAtRegularInterval(string timeIntervalString,
          string timeOffsetString)
```

Derive a regular interval time series data set by interpolation of the current regular or irregular interval time series data set.

The new time interval is set by **timeIntervalString** which must be a valid HEC time interval string (e.g. "1MIN", "15MIN", "1HOUR", "6HOUR", "1DAY", "1MONTH").

Times in the resultant time series may be shifted (offset) from the regular interval time by the increment specified by `timeOffsetString`. As an example, the offset could be used to shift times from the top of the hour to 6 minutes past the hour. If no offset is used timeOffsetString should be an blank or empty string.

Whether the time series data type is "INST-VAL", "INST-CUM", "PER-AVE", or "PER-CUM" controls how the interpolation is performed. Interpolated values are derived from "INST-VAL" or "INST-CUM" data using linear interpolation. Values are derived from "PER-AVE" data by computing the period average value over the time interval. Values are derived from "PER-CUM" data by computing the period cumulative value over the new time interval

For example, if the original data set is hourly data and the new regular interval data set is to have a six hour time interval:

- The value for "INST-VAL" or "INST-CUM" type data is computed from the linear interpolation of the hourly points bracketing the new six hour time point.
- The value for "PER-AVE" type data is computed from the period average value over the six hour interval.
- The value for "PER-CUM" type data is computed from the accumulated value over the six hour interval.

The treatment of missing value data is also dependent upon data type. Interpolated "INST-VAL" or "INST-CUM" points must be bracketed or coincident with valid (not missing) values in the original time series; otherwise the interpolated values are set as missing. Interpolated "PER-AVE" or "PER-CUM" data must contain all valid values over the interpolation interval; otherwise the interpolated value is set as missing.

Parameters:

`timeIntervalString` – A string specifying the regular time interval for the resultant time series.

`timeOffsetString` – A string specifying the offset of the new time points from the regular interval time. This variable may be an empty string (" ").

`Example:`

```
newTsData =
      tsData.interpolateDataAtRegularInterval(
            "15MIN",
            "  ")
```

Returns: A new regular interval TimeSeriesMath object.

7.15.28 Inverse (1/X) Function

`inverse()`

Derive a new time series or paired data set from the inverse (1/x) of values of the current data set. The inverse value is computed by 1.0 divided by the value of the current data set. If a data value is equal to 0.0, the value in the resultant data set is set to missing. For time series data, if the original value is missing, the value remains missing in the resultant data set.

For paired data sets, use the setCurve method to first select the paired data curve(s).

See also: setCurve().

Parameters: Takes no parameters

`Example:  newDataSet = dataSet.inverse()`

Returns: A HecMath object of the same type as the current object.

7.15.29 Determine if Data is in English Units

`isEnglish()`

Determine if the current time series or paired data set is in English units. The function examines the data set parameter type and units label to establish the unit system.

See also: isMetric(); convertToEnglishUnits().

Parameters: No parameters.

```
Example:  if dataSet.isEnglish() : print "English Units"
```

Returns: Constants.TRUE if the data set units are English, otherwise Constants.FALSE.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the unit system cannot be determined (parameter type and units label undefined).

7.15.30 Determine if Data is in Metric Units

`isMetric()`

Determine if the current time series or paired data set is in Metric (SI) units. The function examines the data set parameter type and units label to establish the unit system.

See also: isEnglish(); convertToMetricUnits().

Parameters: No parameters.

```
Example:  if dataSet.isMetric() : print "SI Units"
```

Returns: Constants.TRUE if the data set units are Metric, otherwise Constants.FALSE.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the unit system cannot be determined (parameter type and units label undefined).

7.15.31 Determine if Computation Stable for Given Muskingum Routing Parameters

```
isMuskingumRoutingStable(integer numberSubreaches,
      floating-point muskingumK,
      floating-point muskingumX)
```

Check for possible instability for the given Muskingum Routing parameters.

Test if the input parameters satisfy the stability criteria:

1/(2(1-x)) <= K/deltaT <= 1/2x

where deltaT = (time series time interval)/numberSubreaches

Parameters:

`numberSubreaches` – integer specifying the number of routing subreaches.

`muskingumK` –floating-point number specifying the Muskingum "K" parameter, in hours.

`muskingumX` - floating-point number specifying the Muskingum "x" parameter, between 0.0 and 0.5 (inclusive).

Example:

```
warning = tsDataSet.isMuskingumRoutingStable(
      reachCount,
      kVal,
      xVal)
if warning :
      print warning
      return
```

Returns: A string if the stability criteria is not met. The string contains a warning message detailing the specific instability problem. Otherwise returns None.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the current time series is not a regular interval time series, or if values for numberSubreaches or muskingumX are invalid.

7.15.32 Last Valid Value's Date and Time

`lastValidDate()`

Find and return the date and time of the last valid (non-missing) value in a time series data set.

Parameters: Takes no parameters

Example: `tsData.lastValidDate()`

Returns: An integer value translatable by HecTime representing the date and time of the last valid time series value.

7.15.33 Last Valid Value in a Time Series

`lastValidValue()`

Find and return the last valid (non-missing) value in a time series data set.

Parameters: Takes no parameters

Example: `tsData.lastValidValue()`

Returns: A floating-point value representing the last valid time series value.

7.15.34 Linear Regression Statistics

LinearRegressionStatistics is a class used to contain the linear regression and other correlation coefficients computed by the "correlationCoefficients" function.

The data members of LinearRegressionStatistics are:

integer	numberValidValues	
floating-point	regressionConstant	- intercept of regression line
floating-point	regressionCoefficient	- slope of regression line
floating-point	determinationCoefficient	
floating-point	standardErrorOfRegression	
floating-point	adjustedDeterminationCoefficient	
floating-point	adjustedStandardErrorOfRegression	

The "toString()" method will produce a multi-line character string that can be used to printout the correlation values and description.

Example:

```
linRegData =
      tsData.correlationCoefficients(otherTsData)
regCoef = linRegData.regressionCoefficient
print linRegData
```

See also: correlationCoefficients().

7.15.35 Natural Log, Base "e" Function

`log()`

Derive a new time series or paired data set from the natural log (log base "e") of values of the current data set. Missing values in the original data set remain missing. Values less than or equal to 0.0 will be set to missing.

For paired data sets, use the setCurve method to first select the paired data curve(s).

See also: log10(), setCurve().

Parameters: Takes no parameters

`Example: newDataSet = dataSet.log()`

Returns: A new HecMath object of the same type as the current object.

7.15.36 Log Base 10 Function

`log10()`

Derive a new time series or paired data set from the log base 10 of values of the current data set. Missing values in the original data set remain missing. Values less than or equal to 0.0 will be set to missing.

For paired data sets, use the setCurve method to first select the paired data curve(s).

See also: log(), setCurve().

Parameters: Takes no parameters

`Example: newDataSet = dataSet.log10()`

Returns: A new HecMath object of the same type as the current object.

7.15.37 Maximum Value in a Time Series

`max()`

Find and return the maximum value of the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

`Example: maxVal = tsData.max()`

Returns: A floating-point value representing the maximum value of the current time series.

7.15.38 Maximum Value's Date and Time

`maxDate()`

Find and return the date and time of the maximum value for the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

`Example: maxDateTime = tsData.maxDate()`

Returns: An integer value translatable by HecTime representing the date and time of the maximum time series value.

7.15.39 Mean Time Series Value

`mean()`

Compute the mean value of the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

`Example: meanVal = tsData.mean()`

Returns: A floating-point value representing the mean value of the current time series.

7.15.40 Merge Paired Data Sets

`mergePairedData``(PairedDataMath pdData)`

Merge the current paired data set with the paired data set **pdData**. The resultant paired data set includes all the paired data curves from the current data set. Depending upon a previous use of the setCurveMethod on **pdData,** a single selected paired data curve or all curves from **pdData** are appended to the merged data set. The x-values for the two paired data sets must match exactly.

See also: setCurve().

Parameters:

> **`pdData`** – A paired data set with x-ordinates matching those of the current data set.

`Example: mergedCurve = curve.mergePairedData(anotherCurve)`

Returns: A new PairedDataMath object.

7.15.41 Merge Two Time Series Data Sets

`mergeTimeSeries(TimeSeriesMath tsData)`

Merge data from the current time series data set with the time series data set **tsData**. The resultant time series data set includes all the data points in the two time series, except where the data points occur at the same time. When data points from the two data sets are coincident in time, valid values in the current time series take precedence over valid values from **tsData**. However, if a coincident point is set to missing in the current time series data set, a valid value from **tsData** will be used for time in the resultant data set. If the values are missing for both data sets, the value is missing in the resultant data set.

The data sets for merging may have either regular or irregular time interval time series data. The data sets are tested to determine if they both have the same regular time interval. If not, the resultant data set is typed as an irregular interval data set.

Parameters:

`tsData` – A time series data set for merging with the current time series data set.

`Example: tsMerged = tsData.mergeTimeSeries(otherTsData)`

Returns: A new TimeSeriesMath object.

7.15.42 Minimum Value in a Time Series

`min()`

Find and return the minimum value of the current a time series data set. Missing values are ignored.

Parameters: Takes no parameters

`Example: minVal = tsData.min()`

Returns: A floating-point value representing the minimum value of the current time series.

7.15.43 Minimum Value's Date and Time

`minDate()`

Find and return the date and time of the minimum value for the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

`Example: minDateTime = tsData.minDate()`

Returns: An integer value translatable by HecTime representing the date and time of the minimum time series value.

7.15.44 Modified Puls or Working R&D Routing Function

```
modifiedPulsRouting(TimeSeriesMath tsFlow,
      integer numberSubreaches,
      floating-point muskingumX )
```

The current data set is a paired data set containing the storage-discharge table for Puls routing, where the x-values are storage and the y-values are discharge. The function derives a new time series data set from the Modified Puls or Working R&D routing of the time series data set **tsFlow**. **numberSubreaches** is the number of routing subreaches.

The Working R&D method provides a means of including the effects of inflow on reach storage by use of the Muskingum "x" wedge coefficient. The Working R&D method is activated in the computation if **muskingumX** is greater than 0.0. However, **muskingumX** cannot be greater that 0.5.

Parameters:

tsFlow – A regular interval time series data set for routing.

numberSubreaches – Number of routing subreaches.

muskingumX - Muskingum "X" parameter, between 0.0 and 0.5 (inclusive). Enter 0.0 to route by the Modified Puls method, or a value greater than 0.0 to apply the Working R&D.

Example:

```
routedFlow =
      storDichareCurve.modifiedPulsRouting(
            tsFlow,
            reachCount,
            coefficient)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws an hec.hecmath.HecMathException if the tsMath is not a regular interval time series; if muskingumX is less than 0.0 or greater than 0.5; if the current paired data set does not have both ascending x and y values.

7.15.45 Multiple Linear Regression Coefficients

```
multipleLinearRegression( sequence tsDataSequence,
      floating-point minimumLimit,
      floating-point maximumLimit )
```

Compute the multiple linear regression coefficients between the current time series data set and the array of independent time series data sets in **tsDataSequence**. The function stores the regression coefficients in a new paired data set. This paired data set may be used with the multipleLinearRegression function to derive a new estimated time series data set.

For the general linear regression equation, a dependent variable, Y, may be computed from a set independent variables, Xn:

Y = B0 + B1*X1 + B2*X2 + B3*X3

where Bn are linear regression coefficients.

For time series data sets, an estimate of the original time series data set values may be computed from a set of independent time series data sets using regression coefficients such that:

TsEstimate(t) = B0 + B1*TS1(t) + B2*TS2(t) + … + Bn*TSn(t)

where Bn are the set of regression coefficients and TSn are the time series data sets contained in **tsDataSequence**.

The parameters **minimumLimit** and **maximumLimit** may be used to exclude out of range values in the current time series data set from the regression determination. **minimumLimit** or **maximumLimit** may be entered as "Constants.UNDEFINED" to ignore the minimum or maximum value check.

See also: applyMultipleLinearRegression().

Parameters:

`tsDataSequence` – sequence of TimeSeriesMath objects, which form the independent variables in the regression equation. Must all be regular interval and have the same time interval.

`minimumLimit` – A floating-point value. Values in the current time series exceeding minimumLimit are excluded from the regression analysis. Set to Constants.UNDEFINED to ignore this option.

`maximumLimit` – A floating-point value. Values in the current time series exceeding maximumLimit are excluded from the regression analysis. Set to Constants.UNDEFINED to ignore this option.

Example:

```
regression = tsFlow.multipleLinearRegression (
      [tsUpstrFlow1, tsUpstrFlow2, tsUpstrFlow3],
      0.,
      100000.)
```

Returns: A new PairedDataMath object containing the computed regression coefficients.

Generated Exceptions: `Throws an` hec.hecmath.HecMathException if the current data set and the data sets in tsDataSequence are not regular interval time series data sets with the same interval time.

7.15.46 Multiply by a Constant

```
multiply(floating-point constant)
```

Multiply the value **constant** to all valid values in the current time series or paired data set. For time series data, missing values are kept as missing.

For paired data, **constant** multiplies the y-values only. Use the setCurveMethod to first select the paired data curve(s).

See also: multiply(TimeSeriesMath tsData); setCurve()**.**

Parameters:

`constant` - A floating-point precision value.

Example: `newDataSet = dataSet.multiply(1.5)`

Returns: A new HecMath object of the same type as the current object.

7.15.47 Multiply by a Data Set

```
multiply(TimeSeriesMath tsData)
```

Multiply valid values in the current data set by the corresponding values in the data set **tsData**. Both data sets must be time series data set.

When multiplying one time series data set to another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be multiplied. Times in the current time series data set that cannot be matched with times in the second data set are set to missing. Values in the current data set that are missing are kept as missing. Either or both data sets may be regular or irregular interval time series.

See also: multiply(floating-point constant)**.**

Parameters:

`tsData` - A time series data set.

Example: `newTsData = tsData.multiply(otherTsData)`

Returns: A new TimeSeriesMath object.

7.15.48 Muskingum Hydrologic Routing Function

```
muskingumRouting( integer numberSubreaches,
      floating-point muskingumK,
      floating-point muskingumX)
```

Route the current regular interval time series data set by the Muskingum Routing method. The current data set must be a regular interval time series data set. **muskingumK** is the Muskingum "K" parameter, in hours, and **muskingumX** is the Muskingum "x" parameter. **muskingumX** cannot be less than 0.0 or greater than 0.5.

The set of Muskingum routing parameters may potentially produce numerical instabilities in the routed time series. Use the function isMuskingumRoutingStable() to test if the Muskingum routing parameters may potentially have instabilities.

See also: isMuskingumRoutingStable().

Parameters:

`numberSubreaches` – An integer specifying the number of routing subreaches.

`muskingumK` – A floating-point number specifying the Muskingum "K" parameter in hours.

`muskingumX` – A floating-point number specifying the Muskingum "x" parameter, between 0.0 and 0.5

`Example:`

```
routedFlows = tsFlows.muskingumRouting(reachCount, K, x)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: `Throws an` hec.hecmath.HecMathException if the current time series is not a regular interval time series; if **muskingumX** is less than 0.0 or greater than 0.5.

7.15.49 Number of Missing Values in a Time Series

`numberMissingValues()`

Count and return the number of missing values in the current time series data set.

Parameters: Takes no parameters

`Example: missingCount = tsData.numberMissingValues()`

Returns: An integer of the count of missing time series values.

7.15.50 Number of Valid Values in a Time Series

`numberValidValues()`

Count and return the number of valid values in the current time series data set.

Parameters: Takes no parameters

`Example: validCount = tsData.numberValidValues()`

Returns: An integer of the count of valid (non-missing) time series values.

7.15.51 Olympic Smoothing

```
olympicSmoothing( integer numberToAverageOver,
      boolean onlyValidValues,
      boolean useReduced)
```

Derive a new time series from the Olympic smoothing of **numberToAverageOver** values in the current time series. **numberToAverageOver** must be and odd integer and greater than. Similar to centered moving average smoothing, except that the minimum and maximum values over the averaging interval are excluded from the computation.

If **onlyValidValues** is set to true, then if any values in the averaging interval are missing, the point in the resultant time series is set to missing. If **onlyValidValues** is set to false and there are missing values in the averaging interval, a smoothed point is still computed using the remaining valid values in the interval. If there are no valid values in the averaging interval, the point in the resultant time series is set to missing.

If **useReduced** is set to true, then moving average values can be still be computed at the beginning and end of the time series even if there are less than **numberToAverageOver** values in the interval. If **useReduced** is set to false, then the first and last numberToAverageOver/2 points of the resultant time series are set to missing.

Parameters:

numberToAverageOver – An integer specifying the number of values to average over for computing the smoothed time series. Must be an odd integer greater than 2.

onlyValidValues – Either Constants.TRUE, or Constants.FALSE, specifying whether all values in the averaging interval must be valid for the computed point in the resultant time series to be valid.

useReduced - Either Constants.TRUE, or Constants.FALSE, specifying whether to allow points at the beginning and end of the smoothed time series to be computed from a reduced (less than numberToAverageOver) number of values. Otherwise, set the first and last numberToAverageOver/2 points of the new time series to missing.

Example:

```
avgData = tsData.olympicSmoothing(
      5,
      1)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws a HecMathException if the numberToAverageOver is less than 3 or not odd.

7.15.52 Period Constants Generation

`periodConstants(TimeSeriesMath tsData)`

Derive a new time series data set by applying values in the current time series data set to the times defined by the time series data set **tsData**. Both time series data sets may be regular or irregular interval. Values in a new time series are set according to:

$$ts1(j) \leq tsnew(i) < ts1(j+1)\,, \qquad TSNEW(i) = TS1(j)$$

where ts1 is the time in the current time series, TS1 is the value in the current time series, tsnew is the time in the new time series, TSNEW is the value in the new time series.

If times in the new time series precede the first data point in the current time series, the value for these times is set to missing. If times in the new time series occur after the last data point in the current time series, the value for these times is set to the value of the last point in the current time series. Figure 7.7 shows interpolation of values with the periodConstants function.

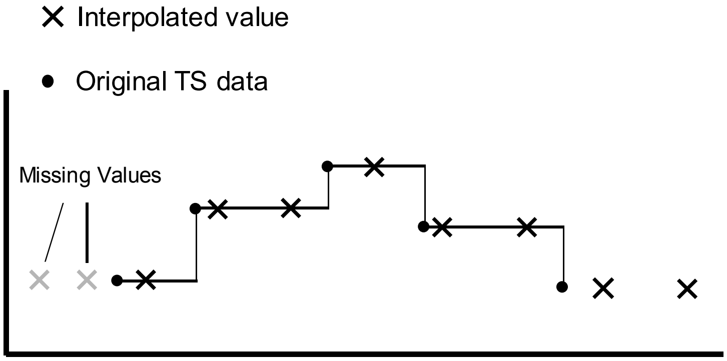


Figure 7.7 Interpolation of Time Series Values Using Period Constants function

Parameters:

`tsData` – A regular or irregular interval time series data set.

`Example:`

```
tsConstants = tsValues.periodConstants(tsData)
```

Returns: A new TimeSeriesMath object.

7.15.53 Polynomial Transformation

`polynomialTransformation(TimeSeriesMath tsData)`

Compute a polynomial transformation of a regular or irregular interval time series data set, **tsData**, using the polynomial coefficients stored in the current paired data set. Missing values in **tsData** remain missing in the resultant data set.

A new time series can be computed from an existing time series with the polynomial expression:

$$TS2\ (t) = B1^{*}\ TS1(t) + B2^{*}\ TS1(t)^{2} + ... + Bn^{*}\ TS1(t)^{n}$$

where Bn are the polynomial coefficients for term "n."

Values for the polynomial coefficients are stored in the x-values of the current paired data set. Before the above equation is applied, values in the input time series are adjusted by subtracting off the paired data "datum" value if defined. The units label and parameter type for the resultant time series are copied from the current paired data set x-units and parameter type.

See also: polynomialTransformationWithIntegral().

Parameters:

`tsData` – A regular or irregular interval time series data set.

`Example: tsXform = pdCoef.polynomialTransformation(tsData)`

Returns: A new TimeSeriesMath object.

7.15.54 Polynomial Transformation with Integral

`polynomialTransformationWithIntegral(TimeSeriesMath tsData)`

Compute a polynomial transformation with integral of a regular or irregular interval time series data set, **tsData**, using the polynomial coefficients stored in the current paired data set. Missing values in **tsData** remain missing in the resultant data set.

This function is similar to the polynomialTranformation method, and the same set of polynomial coefficients are used. The equation for the polynomial transform is modified so that the transform of **tsData** is computed from the integral of the polynomial coefficients:

$$TS2\ (t) = B1^{*}\ TS1(t)^{2}/2\ \ + B2^{*}TS1(t)^{3}/3 + ... + Bn^{*}\ TS1(t)^{n+1}/(n+1)$$

where Bn are the polynomial coefficients for term "n."

Values for the polynomial coefficients are stored in the x-values of the current paired data set. Before the above equation is applied, values in the input time series are adjusted by subtracting off the paired data "datum" value if defined. The units label and parameter type for the resultant time series are copied from the current paired data set x-units and parameter type.

See also: polynomialTransformation().

Parameters:

`tsData` – A regular or irregular interval time series data set.

Example:

```
tsXform =
    pdCoef.polynomialTransformationWithIntegral(tsData)
```

Returns: A new TimeSeriesMath object.

7.15.55 Rating Table Interpolation

`ratingTableInterpolation``(TimeSeriesMath tsData)`

Transform/interpolate values in the time series data set **tsData** using the rating table x-y values stored in the current paired data set. For example, you can use the function to transform a time series of stage to a time series of flow using a stage-flow rating table. **tsData** may be a regular or irregular time interval data set. Missing values in **tsData** are kept missing in the resultant data set.

Create the paired data set with the rating table option to set values for "**`datum", "shift",`** and "**`offset."`** By default these values are 0.0. The shift is added to and the datum subtracted from all input time series values. If the rating table is Log-Log, the table x-values are adjusted by subtracting the offset.

Units and parameter type in resultant time series data set are defined by the y-units label and parameter type of the current paired data set. All other names and labels are copied over from **`tsData`**.

See also: reverseRatingTableInterpolation().

Parameters:

`tsData` – A regular or irregular interval TimeSeriesMath object.

Example:

```
tsFlow =
    stageFlowCurve.ratingTableInterpolation(tsStage)
```

Returns: A new TimeSeriesMath object.

7.15.56 Reverse Rating Table Interpolation

```
reverseRatingTableInterpolation(TimeSeriesMath tsData)
```

Transform/interpolate values in the time series data set **tsData** using the reverse of the rating table stored in the current paired data set. For example, the function may be used to transform a time series of flow to a time series of stage using a stage-flow rating table. **tsData** may be a regular or irregular time interval data set. Missing values in **tsData** are kept missing in the resultant data set.

The paired data set should be created with the rating table option to set values for "`datum", "shift",` and "`offset."` By default, these values are 0.0. The shift is subtracted from, and the datum added to all input time series values. If the rating table is Log-Log, the table x-values are adjusted by subtracting the offset. Refer to the ratingTableInterpolation() description for comparison to this function.

Units and parameter type in resultant time series data set are defined by the x-units label and parameter type of the current paired data set. All other names and labels are copied over from the `tsData`.

See also: ratingTableInterpolation.

Parameters:

`tsData` – A regular or irregular interval TimeSeriesMath object.

Example:

```
tsStage =
    stageFlowCurve.reverseRatingTableInterpolation(
      tsFlow)
```

Returns: A new TimeSeriesMath object.

7.15.57 Round to Nearest Whole Number

```
round()
```

Rounds values in a time series or paired data set to the nearest whole number.

The function rounds up the decimal portion of a number if equal to or greater than .5 and rounds down decimal values less than .5. For example:

10.5	is rounded to 11.
10.499	is rounded to 10.

The x-values in paired data sets are unaffected by the function, only the y-value data are rounded. For time series data sets, missing values are kept missing.

For paired data sets, use the setCurve() method to first select the paired data curve(s).

See also: roundOff();

truncate();

setCurve().

Parameters: Takes no parameters

```
Example:  roundedData = dataSet.round()
```

Returns: A new HecMath object of the same type as the current object.

7.15.58 Round Off to Specified Precision

```
roundOff(integer significantDigits, integer powerOfTensPlace)
```

Round values in a time series or paired data set to a specified number of significant digits and/or power of tens place. For the power of tens place, -1 specifies rounding to one-tenth (0.1), while +2 rounds to the hundreds (100). For example:

1234.123456 will round to:

1230.0	for number of significant digits = 3, power of tens place = -1
1234.1	for number of significant digits = 6, power of tens place = -1
1234	for number of significant digits = 6, power of tens place = 0
1230	for number of significant digits = 6, power of tens place = 1

The x-values in paired data sets are unaffected by the function, only the y-value data are rounded. For time series data sets, missing values are kept missing.

For paired data sets, use the setCurve() method to first select the paired data curve(s).

See also: round();

truncate();

setCurve().

Parameters:

`significantDigits` – An integer specifying the number of significant digits to use in the rounding.

`powerOfTensPlace` – An integer specifying the power of tens place to use in the rounding.

```
Example:  roundedData = dataSet.roundOff(5, -2)
```

Returns: A new HecMath object of the same type as the current object.

7.15.59 Screen for Erroneous Values Based on Forward Moving Average

```
screenWithForwardMovingAverage(integer numberToAverageOver,
      floating-point changeLimit,
      boolean setInvalidToMissingValue,
      string qualityFlagForInvalidValue)
```

Screen the current time series data set for possible erroneous values based on the deviation from the forward moving average over **numberToAverageOver** values computed at the previous point. If the deviation from the moving average is greater than **changeLimit**, the value fails the screening test. Data values failing the screening test are assigned a quality flag and/or are set to missing.

Missing values and values failing the screening test are not counted in the moving average and the divisor of the average is less one for each such value. At least 2 values must be defined in the moving average else the moving average is undefined and value being examined is screened acceptable.

If **setInvalidToMissingValue** is true, values failing the screening test are set to missing.

If **qualityFlagForInvalidValue** is set to a character or string recognized as a valid quality flag, the quality flag will be set for tested values. If there is no previously existing quality available for the time series, the quality flag array will be created for the time series. Values failing the quality test are set to the user specified quality flag for invalid values. If there is existing quality data and the time series value passes the quality test, the existing quality flag for the points is unchanged. If there was no previously existing quality and the time series value passes the quality test, the quality flag for the point is set to "Okay."

The acceptable values for **qualityFlagForInvalidValue** strings are: "M" or "Missing", "R" or "Rejected", "Q" or "Questionable." A blank string (" ") is entered to disable the setting of the quality flag.

For the example,

```
resultantDataSet = dataSet.screenWithForwardMovingAverage (
      16, 100., Constants.TRUE, "R" )
```

the forward moving average will be computed over 16 values, values deviating from the moving average by more than 100.0 will be set to missing and flagged as rejected.

Parameters:

`numberToAverageOver` – An integer specifying the number of averaging values. Must be at least 2.

`changeLimit` – A floating-point number specifying the maximum change allowed in the tested value from the forward moving average value.

`setInvalidToMissingValue` – Either Constants.TRUE, or Constants.FALSE, specifying whether time series values failing the screening test are set to the "Missing" value.

`qualityFlagForInvalidValue` - A string representing the quality flag setting for values failing the screening test. The accepted character strings are: "M" or "Missing", "R" or "Rejected", "Q" or "Questionable." An empty string (" ") is entered to disable the setting of the quality flag.

Example:

```
screenedData = tsData.screenWithForwardMovingAverage(
      16,
      100.,
      Constants.TRUE,
      "R")
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws a HecMathException if `numberToAverageOver` is less than 2; if an unrecognized quality flag is entered for `qualityFlagForInvalidValue` or if `setInvalidToMissingValue` is false and `qualityFlagForInvalidValue` is blank (no action would occur).

7.15.60 Screen for Erroneous Values Based on Maximum/Minimum Range

```
screenWithMaxMin(floating-point minValueLimit,
      floating-point maxValueLimit,
      floating-point changeLimit,
      boolean setInvalidToMissingValue,
      string qualityFlagForInvalidValue)
```

Flag values in a time series data set exceeding minimum and maximum limit values or maximum change limit.

Values in the time series are screened for quality. Values below **minValueLimit** or above **maxValueLimit** or with a change from the previous time series value greater than **changeLimit** fail the screening test. The maximum change comparison is done only when consecutive values are not flagged.

If **setInvalidToMissingValue** is set to true, values failing the screening test are set to the "Missing" value.

If **qualityFlagForInvalidValue** is set to a character or string recognized as a valid quality flag, the quality flag will be set for tested values. If there is no previously existing quality available for the time series, the quality flag array will be created for the time series. Values failing the quality test are set to the user specified quality flag for invalid values. If there is existing quality data and the time series value passes the quality test, the existing quality flag for the points is unchanged. If there was no previously existing quality and the time series value passes the quality test, the quality flag for the point is set to "Okay."

For example,

> resultantDataSet = dataSet.screenWithMaxMin (0.0, 1000., 100., Constants.FALSE, "R")

time series values less than 0.0, or greater than 1000., or with a change from a previous point greater than 100 will be flagged as "Rejected." Flagged points however will not be set to the "Missing" value.

Parameters:

> `minValueLimit` – A floating-point number specifying the minimum valid value limit.
>
> `maxValueLimit` - A floating-point number specifying the maximum valid value limit.
>
> `changeLimit` - A floating-point number specifying the maximum change allowed in the tested value from the previous time series value.
>
> `setInvalidToMissingValue` – Either Constants.TRUE, or Constants.FALSE, specifying whether time series values failing the screening test are set to the "Missing" value.
>
> `qualityFlagForInvalidValue` - A string representing the quality flag setting for values failing the screening test. The accepted character strings are: "M" or "Missing", "R" or "Rejected", "Q" or "Questionable." An empty string (" ") is entered to disable the setting of the quality flag.

Example:

```
screenedData = tsData.screenWithMaxMin(
        0.,
        1000.,
        100.,
        Constants.FALSE,
        "R")
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws a HecMathException if an unrecognized quality flag is entered for `qualityFlagForInvalidValue` or if `setInvalidToMissingValue` is false and `qualityFlagForInvalidValue` is blank (no action would occur).

7.15.61 Select a Paired Data Curve by Curve Label

`setCurve(string curveName)`

Select, by curve label, the paired data curve for performing subsequent arithmetic operations or math functions. By default, a paired data set loaded from file has all curves selected.

A paired data set may contain more than one set of y-values. However, a user may wish to modify only one curve of the data set. For example, using the function `".add( 2.0 )"` would by default add 2.0 to all y-values for all curves. The setCurve() call may be used to limit the operation to just one selected set of y-values.

The function searches the paired data set list of curve labels for a match to **curveName**. If a match is found, that curve is set as the selected curve.

See also: setCurve(integer curveNumber).

`Example: damageCurve.setCurve("RESIDENTIAL")`

Parameters:

`curveName` – The curve label (a string) to set as the selected curve.

Returns: Nothing.

Generated Exceptions: Throws a HecMathException – if curveName is not found in the paired data set curve labels.

7.15.62 Select a Paired Data Curve by Curve Number

`setCurve(integer curveNumber)`

Select, by curve number, the paired data curve for performing subsequent arithmetic operations or math functions. By default, a paired data set loaded from file has all curves selected.

A paired data set may contain more than one set of y-values. However, a user may wish to modify only one curve of the data set. For example, using the function `".add( 2.0 )"` would by default add 2.0 to all y-values for all curves. The setCurve() call can be used to limit the operation to just one selected set of y-values. The function sets a curve index internal to the paired data set. The option is to select one curve or all curves.

Curve numbering begins with "0." If a paired data set has two curves, the first curve is selected by, "setCurve(0)." To select the second curve, use "setCurve(1)."

All curves in a paired data set are selected by setting **curveNumber** to -1.

See also: setCurve(String curveName).

Parameters:

`curveNumber` – An integer specifying the curve to set as the selected curve. Curve numbering begins with 0. Set to –1 to select all curves.

`Example:  ruleCurve.setCurve(-1)`

Returns: Nothing.

7.15.63 Set Data Container

`setData``(hec.io.DataContainer container)`

Sets the data container for the current data set. For time series data sets, this is a **hec.io.TimeSeriesContainer**. For paired data sets, container should be a **hec.io.PairedDataContainer**. Containers are generated by some of the other functions.

The **hec.io.DataContainer** class and the **hec.io.TimeSeriesContainer** and the **hec.io.PairedDataContainer** subclasses contain the time series and paired data values.

Parameters:

`container` – A hec.io.TimeSeriesContainer for time series data sets, or a hec.io.PairedDataContainer for paired data sets.

`Example:  dataSet.setData(TSContainer)`

Returns: Nothing.

Generated Exceptions: Throws a HecMathException if container is not of type **hec.io.TimeSeriesContainer** for time series data sets or not of type **hec.io.PairedDataContainer** for paired data sets.

7.15.64 Set Location Name for Data Set

`setLocation``(String locationName)`

Set the location name for a data set, which changes the B-Part of the HEC-DSS pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

Parameters:

`locationName` – A string specifying the new location name for the data set.

`Example:  dataSet.setLocation("OAKVILLE")`

Returns: Nothing.

7.15.65 Set Parameter for Data Set

setParameterPart(String parameterName)

Set the parameter name for a data set, which changes the C-Part of the HEC-DSS pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

Parameters:

parameterName – A string specifying the new parameter name for the data set.

Example: dataSet.setParameterPart("ELEV")

Returns: Nothing.

7.15.66 Set Pathname for Data Set

setPathname(String pathname)

Set the pathname for a data set to the specified pathname. Subsequent operations using the data set such as getData() or DSSFile.write() will use or reflect the new pathname..

Parameters:

pathname – A string specifying the new pathname for the data set.

Example: dataSet.setPathname("//OAKVILLE/STAGE//1HOUR/OBS/")

Returns: Nothing.

7.15.67 Set Time Interval for Data Set

setTimeInterval(String interval)

Set the time interval for a data set, which changes the E-Part of the pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

Parameters:

interval – A string specifying the new interval for the data set.

Example: dataSet.setTimeInterval("1HOUR")

Returns: Nothing.

7.15.68 Set Data Type for Time Series Data Set

`setType``(string typeString)`

Set the data for a time series data set.

Parameters:

`typeString` – A string specifying the data type for the data set. This should be "INST-CUM", "INST-VAL", "PER-AVER" or "PER-CUM"

`Example: dataSet.setType("PER-AVER")`

Returns: Nothing.

7.15.69 Set Units Label for Data Set

`setUnits``(String unitsString)`

Set the units label for a data set. For a paired data set, the call sets the y-units label.**Parameters:**

`unitsString` – A string specifying the units label for the data set.

`Example: dataSet.setUnits("CFS")`

Returns: Nothing.

7.15.70 Set Version Name for Data Set

`setVersion``(String versionName)`

Set the version name for a data set, which changes the F-Part of the pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

Parameters:

`version` – A string specifying the new location for the data set.

`Example: dataSet.setVersion("OBSERVED")`

Returns: Nothing.

7.15.71 Set Watershed Name for Data Set

`setWatershed``(String watershedName)`

Set the watershed (or river) name for a data set, which changes the A-Part of the pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

Parameters:

`watershedName` – A string specifying the new watershed name for the data set.

`Example: dataSet.setWatershed(“OAK RIVER”)`

Returns: Nothing.

7.15.72 Shift Adjustment of Time Series Data

`shiftAdjustment``(TimeSeriesMath tsData)`

Derive a new time series data set by linear interpolation of values in the current time series data set at the times defined by the time series data set **tsData**. If times in the new time series precede the first data point in the current time series, the value for these times is set to 0.0. If times in the new time series occur after the last data point in the current time series, the value for these times is set to the value of the last point in the current time series. Interpolation of values with the **shiftAdjustment** function is shown in Figure 7.8.

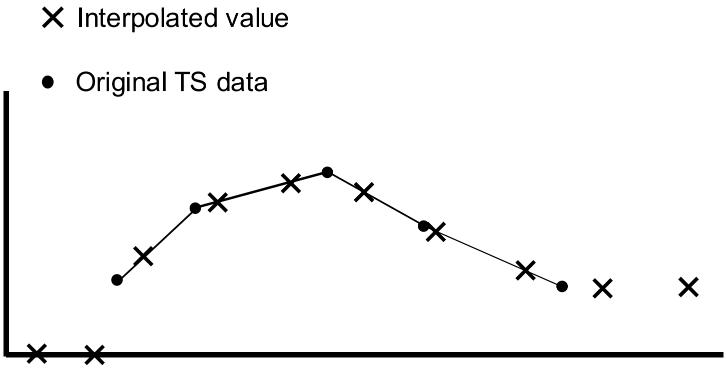


Figure 7.8 Interpolation of Time Series Values using Shift Adjustment Function

Both time series data sets may be regular or irregular interval. Interpolated points must be bracketed or coincident with valid (not missing) values in the original time series, otherwise the values are set as missing.

Parameters:

`tsData` – A regular or irregular interval time series data set.

Example:

```
      tsInterp = tsValues.shiftAdjustment(tsData)
```

Returns: A new TimeSeriesMath object.

7.15.73 Shift Time Series in Time

`shiftInTime(string timeShiftString)`

Shift the times in the current time series data set by the amount specified with **timeShiftString**. The data set may be regular or irregular interval time series data. Data set values are unchanged.

timeShiftString has the form "nT", where "n" is an integer number and "T" is "M"(inute), "H"(our), "D"(ay), "W"(eek), "Mo"(nth),or "Y"(ear). Only the first character is significant for "T", except for "Month", which requires at least two characters.

Parameters:

> **timeShiftString** – A string specifying the time increment to shift times in the current time series data set.

Example: `TsShifted = tsData.shiftInTime("3H")`

Returns: A new TimeSeriesMath object.

7.15.74 Sine Trigonometric Function

`sin()`

Derive a new time series or paired data set from the sine of values of the current data set. The resultant data set values are in radians. For time series data, missing values are kept as missing.

For paired data sets, use the setCurveMethod to first select the paired data curve(s).

See also: setCurve().

Parameters: Takes no parameters

Example: `newDataSet = dataSet.sin()`

Returns: A new HecMath object of the same type as the current object.

7.15.75 Skew Coefficient

`skewCoefficient()`

Compute the skew coefficient of the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

`Example: skewCoefficient = dataSet.skewCoefficient()`

Returns: A floating-point value representing the skew coefficient of the current time series.

7.15.76 Snap Irregular Times to Nearest Regular Period

```
snapToRegularInterval(string timeIntervalString,

      string timeOffsetString,
      string timeBackwardString,
      string timeForwardString )
```

"Snap" data from the current irregular or regular interval time series to form a new regular interval time series of the specified interval and offset. For example, a time series record from a gauge recorder collects readings 6 minutes past the hour. The function may be used to "snap" or shift the time points to the top of the hour.

The regular interval time of the resultant time series is specified by **timeIntervalString**. **timeIntervalString** is a valid HEC time increment string (e.g. "1MIN", "15MIN", "1HOUR", "6HOUR", "1DAY", "1MONTH").

Times in the resultant time series may be shifted (offset) from the regular interval time by the increment specified by **`timeOffsetString`**. As an example, the offset could be used to shift times from the top of the hour to instead 6 minutes past the hour. Data from the original time series is "snapped" to the regular interval if the time of the data falls within the time window set by the **`timeBackwardString`** and the **`timeForwardString`**. That is, if the new regular interval is at the top of the hour and the time window extends to 9 minutes before the hour and 15 minutes after the hour, an original data point at 0852 would be snapped to the time 0900 while a point at 0916 would be ignored.

`timeOffsetString`, **`timeBackwardString`** and **`timeForwardString`** are time increment strings expressed as "nT", where "n" is an integer number and "T" is one of the time increments: "M(INUTES)", "D(AYS)" or "H(OUR) (characters in the parenthesis are optional). For the example of the previous paragraph, **`timeIntervalString`** would be "1HOUR", **`timeOffsetString`** would be "0M", **`timeBackwardString`** would be "9M" (or "9min") and **`timeForwardString`** would be "15M." A blank string (" ") is equivalent to "0M."

By default values in the resultant regular interval time series data set are set to missing unless matched to times in the current time series data set within the time window tolerance set by **`timeBackwardString`** `and` **`timeForwardString`**.

Parameters:

`timeIntervalString` – A string specifying the regular time interval for the resultant time series.

`timeOffsetString` – A string specifying the offset of the new time points from the regular interval time. This variable may be an empty string (" ") or None.

`timeBackwardString` – A string specifying the time to look backwards from the regular time interval for valid time points.

`timeForwardString` – A string specifying the time to look forward from the regular time interval for valid time points.

`Example:`

```
rtsData = itsData.snapToRegularInterval(
      "1HOUR",
      None,
      "5Min",
      "5Min")
```

Returns: A new regular interval TimeSeriesMath object.

7.15.77 Square Root

`sqrt()`

Derive a new time series or paired data set computed from the square root of values of the current data set. For time series data, missing values are kept as missing. Values less than zero are set to missing.

For paired data sets, use setCurve to first select the paired data curve(s).

See also: setCurve().

Parameters: Takes no parameters

`Example:  newDataSet = dataSet.sqrt()`

Returns: A new HecMath object of the same type as the current object.

7.15.78 Standard Deviation of Time Series

```
standardDeviation()
```

Compute the standard deviation value of the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

```
Example:  stdDev = tsData.standardDeviation()
```

Returns: A floating-point value representing the standard deviation of the current time series.

7.15.79 Straddle Stagger Hydrologic Routing

```
straddleStaggerRouting(integer numberToAverage,
      integer numberToLag,
      integer numberSubreaches )
```

Route the current regular interval time series data set using the Straddle-Stagger hydrologic routing method. **numberToAverage** specifies the number of ordinates to average over (Straddle). **numberToLag** specifies the number ordinates to lag (Stagger). The number of routing subreaches is set by **numberSubreaches**.

Parameters:

`numberToAverage` – An integer specifying the number of ordinates to average over (Straddle).

`numberToLag` – An integer specifying the number of ordinates to lag (Stagger).

`numberSubreaches` – An integer specifying the number of routing subreaches.

Example:

```
tsRouted = tsFlow.straddleStaggerRouting(
      numberAver,
      lag,
      reachCount)
```

Returns: A new TimeSeriesMath object.

7.15.80 Subtract a Constant

`subtract(floating-point constant)`

Subtract the value constant from all valid values in the current time series or paired data set. For time series data, missing values are kept as missing.

For paired data, constant is subtracted from y-values only. Use the setCurve method to first select the paired data curve(s).

See also: subtract(HecMath hecMath); setCurve()

Parameters:

`constant` - A floating-point value.

Example: `newDataSet = dataSet.subtract(5.3)`

Returns: A new HecMath object of the same type as the current object.

7.15.81 Subtract a Data Set

`subtract(TimeSeriesMath tsData)`

Subtract the values in the data set **tsData** from the values in the current data set. Both data sets must be time series data set.

When subtracting one time series data set from another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be subtracted. Times in the current time series data set that cannot be matched with times in the second data set are set missing. Values in the current data set that are missing are kept as missing. Either or both data sets may be regular or irregular interval time series.

See also: subtract(floating-point constant).

Parameters:

`tsData` - A TimeSeriesMath object.

Example: `newDataSet = dataSet.subtract(otherDataSet)`

Returns: A new TimeSeriesMath object.

7.15.82 Successive Differences for Time Series

successiveDifferences()

Derive a new time series from the difference between successive values in the current regular or irregular interval time series data set. The current data must be of type "INST-VAL" or "INST-CUM." A value in the resultant time series is set to missing if either the current or previous value in the current time series is missing (need to have two consecutive valid values). If the data type of the current data set is "INST-CUM" the resultant time series data set is assigned the type "PER-CUM", otherwise the data type does not change.

See also: timeDirivative().

Parameters: Takes no parameters

Example: newTsData = tsData.successiveDifferences()

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws a HecMathException if the current data set is not of type "INST-VAL" or "INST-CUM."

7.15.83 Sum Values in Time Series

sum()

Sum the values of the current time series data set. Missing values are ignored.

Parameters: Takes no parameters

Example: total = tsData.sum()

Returns: A floating-point value representing the sum of all valid values of the current time series.

7.15.84 Tangent Trigonometric Function

tan()

Derive a time series or paired data set computed from the tangent of values of the current data set. For time series data, missing values are kept as missing. If the cosine of the current time series value is zero, the value is set missing.

For paired data sets, use the setCurve method to first select the curve(s).

See also: setCurve().

Example: newDataSet = dataSet.tan()

Parameters: Takes no parameters

Returns: A new HecMath object of the same type as the current object.

7.15.85 Time Derivative (Difference Per Unit Time)

```
timeDerivative()
```

Derive a new time series data set from the successive differences per unit time of the current regular or irregular interval time series data set. For the time "t",

TS2(t) = (TS1(t) – TS1(t-1)) / DT

where DT is the time difference between t and t-1. For the current form of the function, the units of DT are minutes.

A value in the resultant time series is set to missing if either the current or previous value in the original time series is missing (need to have two consecutive valid values). By default, the data type of the resultant time series data set is assigned as "PER-AVER."

See also: successiveDifferences().

Parameters: Takes no parameters

Example: `newTsData = tsData.timeDerivative()`

Returns: A new TimeSeriesMath object.

7.15.86 Transform Time Series to Regular Interval

```
transformTimeSeries(string timeIntervalString,
      string timeOffsetString,
      string functionTypeString )
```

Generate a new regular interval time series data set from the current regular or irregular time series. The new time series is computed having the regular time interval specified by **timeIntervalString** and time offset set by **timeOffsetString**.

Values for the new time series are computed from the original time series data set using one of seven available functions. The function is selected by setting **functionTypeString** to one of the following types:

"INT"	-	Interpolate at end of interval
"MAX"	-	Maximum over interval
"MIN"	-	Minimum over interval
"AVE"	-	Average over interval
"ACC"	-	Accumulation over interval
"ITG"	-	Integration over interval
"NUM"	-	Number of valid data over interval

where "interval" is the interval between time points in the new time series.

The regular interval time of the new time series is specified by **timeIntervalString**. **timeIntervalString** is a valid HEC time increment string (e.g. "1MIN", "15MIN", "1HOUR", "6HOUR", "1DAY", "1MONTH").

Times in the resultant time series may be shifted (offset) from the regular interval time by the increment specified by `timeOffsetString`. As an example, the offset could be used to shift times from the top of the hour to 6 minutes past the hour. Typically no offset is used.

The data type of the original time series data governs how values are interpolated. Data type "INST-VAL" (or "INST-CUM") considers the value to change linearly over the interval from the previous data value to the current data value. Data type "PER-AVER" considers the value to be constant at the current data value over the interval. Data type "PER-CUM" considers the value to increase from 0.0 (at the start of the interval) up to the current value over the interval. Interpolation of the three data types is illustrated in Figure 7.9.

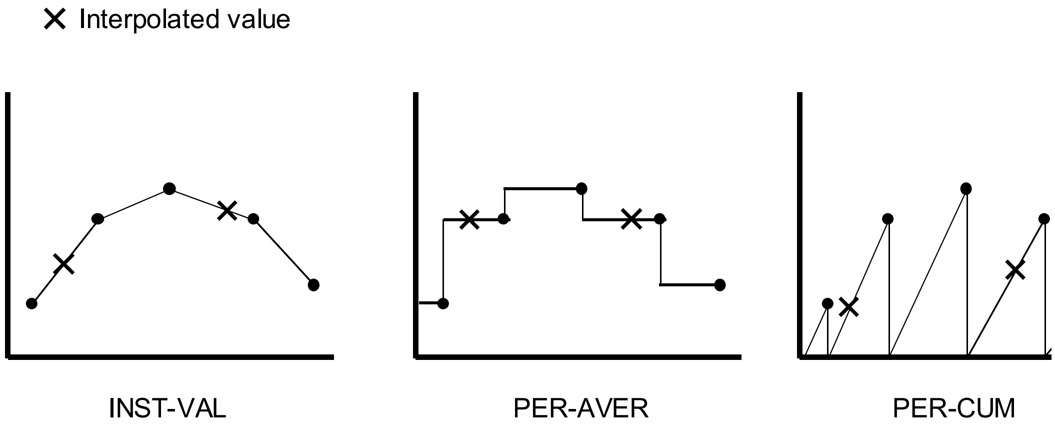


Figure 7.9 Interpolation of "INST-VAL", "PER-AVER" and "PER-CUM" data

How interpolation is performed for a specific data type influences the computation of new time series values for the selected function. For example, if the data type is "INST-VAL", the function "Maximum over interval" is evaluated by: Finding the maximum value of the data points from the original time series that are inclusive in the new time interval. Linearly interpolate values at beginning and ending of the new time interval, and determine if these values represent the maximum over the interval.

Referring to the plots in Figure 7.9, the "Average over interval" function is applied to a time series by integrating the area under the curve between interpolated points and dividing the result by the interval time.

See also: transformTimeSeries(TimeSeriesMath tsData, string functionTypeString)

Parameters:

`timeIntervalString` – A string specifying the regular time interval for the resultant time series.

`timeOffsetString` – A string specifying the offset of the new time points from the regular interval time. This variable may be a blank string (“ ”).

`functionTypeString` – A string specifying the method for computing values for the new time series data set.

Example:

```
newTsData = tsData.transformTimeSeries(
      “1Day”,
      “0M”,
      “AVE”)
```

Returns: A new regular interval TimeSeriesMath object.

7.15.87 Transform Time Series to Irregular Interval

```
transformTimeSeries(TimeSeriesMath tsData,
      string functionTypeString )
```

Generate a new time series data set from the current regular or irregular time series. The times for the new data set are defined by the times in tsData, which may be a regular or irregular time series data set.

Values for the new time series are computed from the original time series data set using one of seven available functions. The function is selected by setting **functionTypeString** to one of the following types:

"INT"	-	Interpolate at end of interval
"MAX"	-	Maximum over interval
"MIN"	-	Minimum over interval
"AVE"	-	Average over interval
"ACC"	-	Accumulation over interval
"ITG"	-	Integration over interval
"NUM"	-	Number of valid data over interval

where “interval” is the interval between time points in the new time series.

The data type of the original time series data governs how values are interpolated. Data type “INST-VAL” (or “INST-CUM”) considers the value to change linearly over the interval from the previous data value to the current value. Data type “PER-AVER” considers the value to be constant at the current value over the interval. Data type “PER-CUM” considers the value to increase from 0.0 (at the start of the interval) up to the current value over the interval. Interpolation of the three data types is illustrated in Figure 7.9.

How interpolation is performed for a specific data type influences the computation of new time series values for the selected function. For example, if the data type is "INST-VAL", the function "Maximum over interval" is evaluated by: Finding the maximum value of the data points from the original time series that are inclusive in the new time interval. Linearly interpolate values at beginning and ending of the new time interval, and determine if these values represent the maximum over the interval.

Referring to the plots in Figure 7.9, the "Average over interval" function is applied to a time series by integrating the area under the curve between interpolated points and dividing the result by the interval time.

See also: transformTimeSeries(string timeIntervalString,

string timeOffsetString, string functionTypeString)

Parameters:

`tsMath` – A TimeSeriesMath object used to define the times for the new data set.

`functionTypeString` – A String specifying the method for computing values for the new time series data set.

`Example:`

```
newTsData = tsValues.transformTimeSeries(
      tsTimeTemplate,
      "MAX")
```

Returns: A new TimeSeriesMath object.

7.15.88 Truncate to Whole Numbers

`truncate()`

Truncates values in a time series or paired data set to the nearest whole number. For example:

10.99 is truncated to 10.

The x-values in paired data sets are unaffected by the function, only the y-value data are truncated. Missing values remain missing.

For paired data, use the setCurve method to first select the curve(s).

See also: setCurve().

Parameters: Takes no parameters

`Example:  newDataSet = dataSet.truncate()`

Returns: A new HecMath object of the same type as the current object.

7.15.89 Two Variable Rating Table Interpolation

```
twoVariableRatingTableInterpolation(
      TimeSeriesMath tsDataX,
      TimeSeriesMath tsDataZ)
```

Derive a new time series data set by using the x-y curves in the current paired data set to perform two-variable rating table interpolation of the time series **tsDataX** and **tsDataZ**. For two-variable rating table interpolation, the current paired data set should have more than one curve (multiple sets of y-values).

As an example, reservoir release is a function of both the gate opening height and reservoir elevation (Figure 7.10). For each gate opening height, there is a reservoir elevation-reservoir release curve, where reservoir elevation is the independent variable (x-values) and reservoir release the dependent variable (y-values) of a paired data set. Each paired data curve has a curve label. In this case, the curve label is assigned the gate opening height. Using the paired data set shown in Figure 7.10, the function may be employed to interpolate time series values of reservoir elevation (**tsDataX**) and gate opening height (**tsDataZ**) to develop a time series of reservoir release.

No extrapolation is performed. If time series values from **tsDataX** or **tsDataZ** are outside the range bounded by the paired data, the new time series value is set to missing. Units and parameter type in the new time series are set to the y-units label and parameter of the current paired data set. All other names and labels are copied over from **tsDataX**.

Times for **tsDataX** and **tsDataZ** must match. Curve labels must be set for curves in the rating table paired data set and must be interpretable as numeric values.

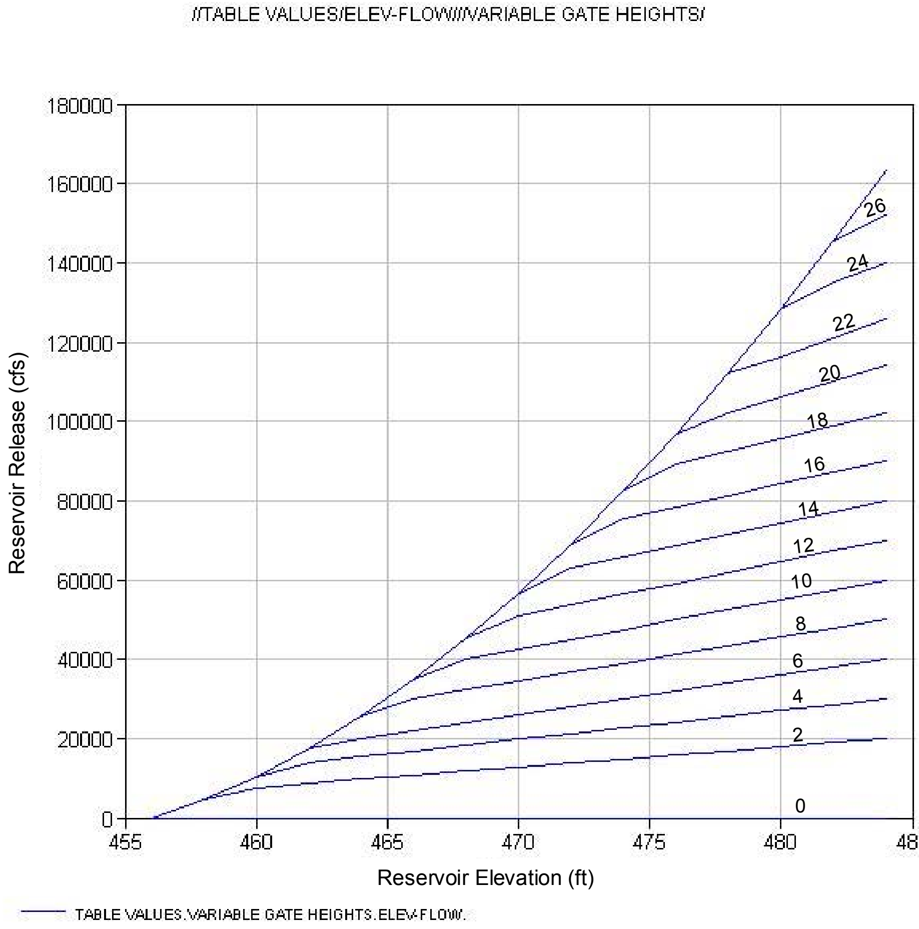


Figure 7.10 Example of two variable rating table paired data, reservoir release as a function of reservoir elevation and gate opening height (curve labels).

Parameters:

`tsDataX` – A regular or irregular interval TimeSeriesMath object, interpreted as x-ordinate values in the two variable interpolation.

`tsDataZ` – A regular or irregular interval TimeSeriesMath object, interpreted as z-ordinate values, (value defined by the paired data curve labels).

```
Example:    tsOutflow =
            gateCurve.twoVariableRatingTableInterpolation(
                  tsElevation,
                  tsGateOpening)
```

Returns: A new TimeSeriesMath object.

Generated Exceptions: Throws a HecMathException if times do not match for **tsDataX** and **tsDataZ**; if the paired data curve labels are blank or cannot be interpreted as number values.